TECHNISCHE UNIVERSITÄT DRESDEN
Fakultät Informatik

Alternative Automata-based Approaches to Probabilistic Model Checking

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der Technischen Universität Dresden
Fakultät Informatik
von **David Müller**
geboren am 08. März 1987
in Dresden

Betreunde Hochschullehrerin/Gutachterin:
Prof. Dr. rer. nat. Christel Baier
Technische Universität Dresden

Gutachter:
Univ.-Prof. Dr. Dr. h.c. Javier Esparza
Technische Universität München

Tag der Verteidigung: 10.Dezember 2018
Dresden im Oktober 2019

Abstract

Probabilistic model checking (PMC) aims to determine the likelihood of a system to meet a specification. In this thesis we consider as models for randomized systems discrete-time Markov chains and discrete-time Markov decision processes (MDPs).

In this thesis we focus on linear temporal logic (LTL) for expressing specifications. The standard approach for non-probabilistic systems translates an LTL formula into a non-deterministic Büchi automaton (NBA) with a single-exponential number of states in the worst case. While in the non-probabilistic setting non-deterministic Büchi automata (NBA) can be used directly, the probabilistic choices are incompatible with the direct use of an NBA. In the standard approach to PMC, the NBA obtained from the LTL formula is determinized, which can lead to a double-exponential number of states in total. There are approaches for Markov chain analysis against LTL with exponential runtime, which motivates the search for non-deterministic automata with restricted forms of non-determinism that make them suitable for PMC. For MDPs, the approach via deterministic automata matches the double-exponential lower bound, but a practical application might benefit from approaches via non-deterministic automata.

We first investigate good-for-games (GFG) automata. In GFG automata one can resolve the non-determinism for a finite prefix without knowing the infinite suffix and still obtain an accepting run for an accepted word. We explain that GFG automata are well-suited for MDP analysis on a theoretic level, but our experiments show that GFG automata constructed by the methods proposed by [HP06] cannot compete with deterministic automata.

We have also researched another form of pseudo-determinism, namely unambiguity, where for every accepted word there is exactly one accepting run. For unambiguous Büchi automata (UBA) it is claimed in the literature that Markov chain analysis for a given UBA can be done in polynomial time. We show that the proposed approach is flawed and present a new polynomial-time approach for PMC of Markov chains against UBA specifications. Instead of identifying single states inducing probability 1 (which may not exists), we identify sets of states inducing probability 1.

Additionally, we examine the new symbolic Muller acceptance described in the Hanoi Omega Automata Format, which we call Emerson-Lei acceptance. It is a positive Boolean formula over unconditional fairness constraints. We present a construction of small deterministic automata using Emerson-Lei acceptance. Deciding, whether an MDP has a positive maximal probability to satisfy an Emerson-Lei acceptance, is NP-complete. This fact has triggered a DPLL-based algorithm for deciding positiveness.

For every approach we perform benchmarks on the LTL-to-automata translation itself as well as on models of the `PRISM` benchmark suite.

Acknowledgments

I am very thankful for the support of Christel Baier. She offered me a very interesting PhD topic, great scientific insights and encouraged me to stick to the topic, even if the results looks bad. She is also a great inspiration in teaching, whether it is to prepare slides, or scripts with a clear and helpful structure. I also want to thank my colleague Joachim Klein, who never declined to discuss automata-theoretic ideas, helped me at programming, and proof-read this thesis. Christel and Joachim complemented each other in great way.

Next, I want to thank our secretaries Karina Wauer, Kerstin Achtruth, Sandy Seifarth, and Kati Michel for their support in bureaucratic stuff. They saved me a lot time, when filling out the reimbursement applications. Of course, I thank also my colleagues Philipp Chrszon, Marcus Daum, Clemens Dubslaff, Daniel Gburek, Simon Jantsch, Lisa Kruse, Linda Leuschner, Sascha Klüppelholz, Steffen Märcker, and Jakob Piribauer for the fruitful discussions and for making the past years so enjoyable.

I also want to express my gratitude to my co-authors Tomáš Babiak, František Blahoudek, Alexandre Duret-Lutz, Jan Křetínský, David Parker and Jan Strejček for the cooperation on the Hanoi-Omega Format and the Emerson-Lei acceptance.

Concerning financing my research, I'm thankful to the german research foundation by funding my work via the research training group QuantLA (1763) and via the project BA 1679/12-1.

At last, I want to show appreciation to my family for their support and believe in me. In particular, my warm-hearted uncle Volkmar taught me programming as a little child and lured in this manner into computer science. I guess, he would have made some funny jokes, if he had seen my thesis.

Contents

1 Introduction

The growing complexity and dependence on computational systems in our every day life renders checking their correctness and safety more complicate. Many errors and pitfalls can be avoided by testing and simulation but both methods are incomplete. Alternatively, formal methods offer an exhaustive system analysis.

One technique for formal verification is interactive theorem proving, see, e.g., [BC04; NK14]. Interactive theorem proving allows the creation of proofs in a user-supported fashion with several benefits such as automatic code generation or automated proof search up to a certain depth.

Model checking belongs to the area of formal methods as well, being an automated technique for the formal analysis of abstract models. In its classical form it decides whether a model of a system satisfies a property. It has been introduced in the early eighties of the last century, with the notable publications [EC80; LP85; CES86], and has spread out into different research lines by now, e.g., analysis of timed automata [AD94; LPY95; BY04] or probabilistic systems [Var85; VW86; CY95; Var99]. For a broad introduction we refer to [BK08].

One of the most basic models is Kripke structures. Kripke structures are labeled graphs where states can be initial. One model of the behavior are paths which are sequences of states. Kripke structures offer a purely non-deterministic behavior: The first state and the successors states of a path are chosen non-deterministically. Every path can be lifted to its trace by taking the labelings instead of the states. The traces of the paths can be seen as the possible behaviors of the Kripke structures.

For specifying properties one usually employs temporal logics. The two most prominent temporal logics are computation tree logic (CTL) and linear temporal logic (LTL).

CTL is a branching time logic, in which one does reason about the computation tree, which is the unfolded behavior of a system. CTL can be checked in polynomial time [CE81; CES86].

LTL focuses on the paths of a model as the semantic of LTL is defined by a set of words. A model satisfies an LTL formula if all the model's traces are contained in the set of words that satisfy the LTL formula, i.e., the model exhibits only behavior characterized by the LTL formula. The typical approach for LTL model checking employs ω-automata. The negated LTL formula is transformed into a (possibly exponential-sized) non-deterministic Büchi automaton (NBA). Then a product is built, in which one can search for behavior that the Kripke structure displays but that is forbidden by the specification.

Prominent model checkers for Kripke structures and LTL are `SPIN` [Hol04] and `nuSMV` [Cim+02]. In practice, the model (and the product) size can become very large

even for simple systems, and thus actual model checking turns unfeasible very fast if the system's model is saved state-by-state in the memory. This problem is called the *state space explosion problem* and led to symbolic reasoning over the models. Instead of analyzing particular states, one does analyze set of states. An important milestone of symbolic reasoning was the introduction of binary decision diagrams (BDDs).

BDDs can serve for a compact representation of the state space of a model or automaton. They have been introduced by [Bry86] and have found their way into several applications, e.g., VLSI design [MT98]. BDDs are essentially a graph representation of Boolean functions, offering a good compromise between computational efficiency and memory consumption. For a general introduction to BDDs we refer to [Bur+92; Weg00].

Another handling of the state explosion problem is offered by bounded model checking (BMC) [Bie+99; Bie+03]. BMC relies on an iterative enumeration of all finite paths up to a certain bound k. If a counterexample for the correctness of the system is found, then this counterexample is returned. If k exceeds a threshold marking that the whole model has been explored and no counterexample can be found anymore, the BMC algorithm returns the correctness of the system. In practice, BMC outperforms BDD-based model checking if a counterexample exists, and is therefore useful for prototyping in particular.

Since the basic model checking focuses on Kripke structures and therefore asks for a binary answer whether the specification is satisfied or not, the need for more expressiveness emerged quite naturally. Probabilistic model checking focuses on Markovian models like Markov chains or Markov decision processes (MDPs for short), thus enabling answers like "The system obeys the specification with a likelihood of 99.95%". Markov chains can be seen as Kripke structures with a pure probabilistic transition structure. If one provides non-deterministic choices to Markov chains, one obtains MDPs. Analogously to games, MDPs are called $1\frac{1}{2}$-player games, where one player resolves the non-deterministic choices, and the other (half) player resolves the probabilistic choices probabilistically.

Despite the probabilistic nature of Markovian models the syntax and semantics of LTL remains equal. However, for CTL a probabilistic counterpart, PCTL, exists. Like CTL, PCTL can be checked efficiently in polynomial time, see, e.g., [HJ94] and there exists model checkers focusing on PCTL (or some of its variants), such as `STORM` [Deh+17] and `MRMC` [Kat+11].

The probabilistic choices of Markovian models blocks the direct application of NBAs in the process of model checking Markovian models against LTL formula. As a resort to this problem, deterministic automata are usually employed.

The translation of LTL to deterministic ω-automata can cause a double-exponential blow-up [KV05; KR11]. This double-exponential blow-up together with the polynomial-time algorithms for building the product and the analysis of it yields an overall double-exponential-time algorithm for the analysis of Markov decision processes (MDP) against LTL. This double-exponential time algorithm matches the lower bound [CY95]. The case is different for Markov chains, where a PSPACE lower bound is known [Var85]. Thus, the double-exponential blow-up over deterministic automata

leaves a complexity gap for Markov chains.

1.1 Literature

1.1.1 Markov Decision Processes

First results in the area of probabilistic model checking for MDPs have been achieved for qualitative PMC, where one wants to prove that a path property holds almost surely for all possible or a single resolution of the non-determinism. Alternatively, one wants to prove that a path property holds with a positive probability for all possible or a single resolution of the non-determinism. Hart, Sharir and Pnueli [HSP83] discussed the termination problem in the qualitative setting for a system of concurrent processes. Pnueli continued this work with Zuck in [PZ86a; PZ86b] and presented a tableau-based approach for LTL. The first results on automata-based approaches for MDP analysis have been published by Vardi [Var85] and Vardi and Wolper [VW86], which was revisited by Courcoubetis and Yannakakis [CY95]. They depend on so-called deterministic-in-the-limit Büchi automata, i.e., automata that behave deterministically after reaching a final state. The construction of a deterministic-in-the-limit Büchi automaton out of an NBA resembles the breakpoint construction [MH84], a multi subset construction, and can cause an exponential blow-up. Thus, the overall approach for MDP analysis for LTL matches the lower 2EXPTIME bound.

Probabilistic model checking suffers also from the state space explosion problem. Here, the concept of BDDs has been transferred to multi-terminal BDDs (MTBDDs), i.e., BDDs representing not only Boolean functions but multi-valued functions [CFZ93; Bah+93] as they can represent matrices with real values between 0 and 1. The probabilistic model checker `PRISM` [Par02; KNP11] supports MTBDDs by the two engines `hybrid` and `mtbdd`. In the `hybrid` engine the result vector containing the probability to obey the specification for every state is stored in an explicit manner, whereas in the `mtbdd` engine the result vector is stored symbolically as well. In the `explicit` engine the state-space of the model is represented state-for-state, as well as the transitions.

Besides the employment of MTBDDs, statistical model checking [LDB10; LV15] mitigates the state space explosion problem by simulating the system and sampling a finite number of runs. These runs are used to provide estimates about the correctness of systems within certain bounds.

Recent developments have lowered the computation time for a certain fragment of LTL to single exponential time, namely $\mathrm{LTL}_{\setminus\Box\mathcal{U}}$ [KV15]. This fragment describes LTL formulas in positive normal form using the operators $\bigcirc, \Diamond, \Box$ and $\mathcal{U}$ with the additional restriction that no $\mathcal{U}$ operator occurs in the scope of a $\Box$ operator. In [Kin17] the fragment $\mathrm{LTL}_{\setminus\Box\mathcal{U}}$ is extended to a fragment called $\mathrm{LTL}_{\mathrm{D}}$, where the restrictions of $\mathrm{LTL}_{\setminus\Box\mathcal{U}}$ are loosened in the following way:

- for a formula $\varphi \mathcal{U} \psi$ occurring inside of a $\Box$ operator, ψ has to contain only $\Diamond$

and $\Box$ as temporal operators,

- in every formula of the form $\varphi \vee \psi$ occurring inside of a $\Box$ operator, at least φ or ψ has to contain only $\Diamond$ and $\Box$ as temporal operators.

Covering full LTL, Sickert, Esparza, Jaax and Křetínský offered a new (double-exponential) construction of deterministic-in-the-limit Büchi automata [Sic+16]. These deterministic-in-the-limit automata are even suitable for quantitative probabilistic model checking. As their implementation `MoChiBa` [SK16] shows, that the efficiency of MDP analysis benefits from the usage of deterministic-in-the-limit Büchi automata.

Another tool for the generation of deterministic-in-the-limit automata is `Seminator` [Bla+17], which takes a non-deterministic automaton with a transition-based generalized Büchi acceptance as input, and transforms it into a deterministic-in-the-limit automaton. However, the resulting automaton may be not suited for quantitative PMC, but only qualitative PMC.

The mentioned work relied on building a single ω-automaton equivalent to the specification. The authors of [Hah+15] consider an alternative. They provide a translation into two automata with the same graph structure, but different acceptance conditions, under-approximating and over-approximating the languages respectively. Accordingly, one product with two acceptance conditions is built, and every maximal end-component is checked, whether it satisfies the two acceptance conditions. If the two result are different, the affected automaton part is refined via breakpoint construction or if necessary a full determinization to deliver exact results. The authors implemented this approach into their probabilistic model checker `IscasMC` [Hah+14] and in its successor `ePMC` [Tur17].

1.1.2 Markov chains

The PSPACE lower bound for the analysis of Markov chains against LTL specifications inspired several algorithms avoiding deterministic automata. Courcoubetis and Yannakakis [CY88; CY95] presented an automata-less Markov chain analysis method, that can be lifted to PSPACE algorithm for qualitative analysis. We give short overview for its quantitative version: To calculate $\Pr_{\mathcal{M}}(\varphi)$, the Markov chain $\mathcal{M}$ and the LTL formula φ are refined to a Markov chain $\mathcal{M}'$ and an LTL formula φ' preserving the probability, i.e., $\Pr_{\mathcal{M}}(\varphi) = \Pr_{\mathcal{M}'}(\varphi')$. The refinement is carried out iteratively until the refined LTL formula does not contain any temporal operator, which then can be easily checked.

In a refinement step one selects a subformula ψ with a temporal operator as top-most operator (in the syntax tree of the formula) and no other temporal operator. Then, this subformula is replaced by a fresh atomic proposition in φ, leading to a new LTL formula φ'. The Markov chain $\mathcal{M}$ is transformed with the usage of computed probabilities for every state of $\mathcal{M}$ and ψ in such a way that $\Pr_{\mathcal{M}}(\varphi) = \Pr_{\mathcal{M}'}(\varphi')$ holds, where we denote the transformed Markov chain as $\mathcal{M}'$. The transformation of $\mathcal{M}$ doubles the state space in the worst case.

The refinement step is repeated until φ' does not contain any temporal operator anymore. As the remaining task one has to calculate $\Pr_{\mathcal{M}'}(\varphi')$ for such a simple formula φ'. This step just amounts to a summation over the initial distribution $\iota(s)$ for every state s satisfying φ' and ι being the initial distribution of $\mathcal{M}'$.

Couvreur, Saheb and Sutre [CSS03] suggested separated automata, a special form of unambiguous automata. For a deeper consideration of [CSS03], in particular a comparison with unambiguous automata, we refer to Section 4.1.6.

Instead of a single automaton, [BRV04] takes a weak alternating ω-automaton $\mathcal{A}$ as input and transforms it into two ω-automata, a so-called full automaton and a local transition system. The local transition system and the Markov chain form a product, and then a state is searched with the help of the full automaton, that counter witnesses $\Pr_{\mathcal{M}}(\mathcal{A}) = 1$.

1.1.3 ω-automata

Non-deterministic Büchi automata have been introduced independently in 1962 by Büchi [Büc62] and Trakhtenbrot [Tra62] as tools for decision problems in the area of mathematical logics.

LTL model checking of Kripke structures is usually done by building an automaton equivalent to the negated specification. Vardi and Wolper [VW86] have suggested a rather simple construction method for NBA equivalent to a given LTL specification.

The constructions for NBA became more elaborate, see [GO01; Bab+12] for an approach via alternating automata, or [Ger+95] for a tableau-based approach.

The competition between the LTL-to-NBA translations has lead to the generation of smaller NBA, yielding an improved starting point for determinization algorithms. The research for determinization algorithms follows two main lines: Safra-based methods [Saf88; Pit07; Sch09] (and their heuristic improvements in [KB06; KB07]) and Muller-Schupp based methods [MS95; KW08; Fog+13; FL15]. A hybrid version has been proposed by Redziejowski [Red12].

The earliest approach [McN66] for determinization which was presented by McNaughton has not been pursued much further, since it is double-exponential in the size of the Büchi automaton and therefore does not match the single exponential lower bound.

Since the class of languages equivalent to a deterministic Büchi automaton (also called DBA-realizable languages) is a strict subset of ω-regular languages (in contrast to NBA), more expressive acceptance conditions exist: Rabin [Rab72], Streett [Str82], parity [Mos84], Muller [Mul63] among others. All acceptance conditions can be seen as a special form of a Muller acceptance, which explicitly represents every possible acceptable set of states being visited infinitely often.

Instead of taking an intermediate step via non-deterministic Büchi automata, direct translations from LTL to deterministic ω-automata have been proposed. Křetínský and Esparza [KE12] offer a direct translation to a deterministic (generalized) Rabin automaton for the LTL fragment where $\Diamond$ and $\Box$ are the only allowed temporal operators. This approach has been extended [Bab+13b; KG13; EK14] until full LTL

has been covered. Very recently at LICS 2018, Esparza, Křetínský and Sickert have provided a Master theorem enabling the decomposition of the language described by an LTL formula into fragments offering a simpler translation to automata than translations for full LTL [EKS18].

Apart from striving to generate small deterministic automata during the construction, post generation algorithms for shrinking the automaton size without changing the accepted language have also been proposed.

Simulation delivers a general-purpose method for shrinking automata. They are based on calculating a simulation relation on the state space that can be used to collapse the states within the same equivalence classes and still obtain an equivalent automaton. However, one cannot achieve a minimal number of state with these techniques in case of ω-automata. Prominent examples are stutter simulations [KB07; MD15], which enable to collapse states that are unnecessary finite repetitions, and bisimulation [Mil80; Par81] identifying equivalent substructures in an automaton.

A very intuitive minimization technique concerns weak deterministic Büchi automata (WDBA), i.e., deterministic Büchi automata where every SCC contains either no accepting state or solely accepting states. This requirement reduces the expressiveness of WDBA to the intersection of DBA-realizable languages and coDBA-realizable languages.[1] Löding [Löd01] established the connection between WDBA and deterministic finite automata over finite words (DFA) and reduced the task of minimizing WDBA to minimizing DFA. We have implemented his minimization technique in the probabilistic model checker `PRISM` and could achieve a reduction in 30 out of 44 cases [Kle+17].

For minimization of a wider range of ω-automata, SAT-based approaches have been proposed recently. Schewe proves the NP-completeness of deciding whether a DBA has a minimal number of states and analogous NP-completeness results for deterministic co-Büchi automata, and deterministic parity automata [Sch10]. Ehlers builds on this result and minimizes DBA with SAT-solvers [Ehl10]. To cover full ω-regularity, Baarir and Duret-Lutz generalized this to Emerson-Lei ω-automata [BD15], which are essentially ω-automata with Muller acceptance expressed in a symbolic fashion.

In 2015 the Hanoi Omega Automata Format was published [Bab+15]. The Hanoi Omega Automata Format is a unified exchange format for ω-automata tools. This format allows a tool-agnostic interoperation, e.g., `PRISM` is now able to use every tool that supports the Hanoi Omega Automata Format (and determinization) for the creation of deterministic automata.

Now we turn to specific types of ω-automata discussed in this thesis. This overview should be seen as a collection of literature to separate our contribution. A deeper literature review will be covered in the specific chapters.

[1] DBA-realizable languages are ω-regular languages for which an equivalent DBA exists. Analogously, coDBA-realizable languages are languages, for which an equivalent deterministic co-Büchi ω-automaton exists.

Good-for-games automata. The concept of good-for-games automata (GFG) has been introduced by Henzinger and Piterman [HP06]. In a good-for-games automaton the resolution of the non-determinism for a finite prefix of an accepted word does not require any look-ahead.

It was introduced for the synthesis of reactive systems where one typically solves 2-player games. Good-for-games automata were offered as a substitute to deterministic automata.

In [HP06] a translation from NBA to GFG parity automata has been presented, which can be incorporated into a translation from LTL to GFG parity automata. The authors of [KS15] have shown the potential of good-for-games by exhibiting a family of good-for-games co-Büchi automata, which are exponentially more succinct than every equivalent deterministic automaton. Still, [HP06] proves that a transformation from NBA to GFG automata can cause an exponential blow-up.

Unambiguous Büchi automata. Unambiguity in a non-deterministic automaton demands that there is exactly one accepting run for every accepted word. The phenomenon has been widely studied in the 1980s. In particular, Stearns and Hunt [SH85] have considered universality ("Is every word an accepted word?") for unambiguous regular grammars (and automata) over finite words and could establish a polynomial time result for deciding universality. Bousquet and Löding [BL10] could prove an analogous result for separated Büchi automata, i.e., automata that are still unambiguous if all states have been set to initial. The first authors who looked into the application of separated automata for Markov chain analysis were Couvreur, Sutre and Saheb [CSS03]. As one can construct a separated automaton equivalent to an LTL formula within exponential time, and the Markov chain analysis against separated automata specifications can be done in polynomial time, the overall time complexity for this approach is single exponential.

Based on [CSS03], Benedikt, Lenhardt, and Worrell [BLW13b; BLW14] proposed a generalization to unambiguous Büchi automata. Their claim, that a polynomial-time analysis of Markov chains under UBA specifications is possible, is true, but their proofs rely on a false assumption and therefore their algorithms work incorrectly. This flaw does not affect the second contribution of [BLW14], namely an algorithm for model checking Markov chains against unambiguous automata over finite words.

Emerson-Lei acceptance. The commonly used Rabin, Streett and parity acceptances can be seen as particular forms of Muller acceptance. Muller acceptance explicitly enumerates every possible set of states (or transitions) that is visited infinitely often in an accepting run. This explicit representation is verbose, and therefore in [Bab+15] a more compact symbolic representation is proposed which we call Emerson-Lei acceptance. The idea of Emerson-Lei acceptance goes back to Emerson and Lei [EL87]. They have considered positive Boolean formulas with atoms stating that a certain atomic proposition should be seen infinitely often or only finitely often. Apart from small syntactical differences, the definitions of Emerson and Lei in

[EL87] and the symbolic acceptance in [Bab+15] agree. The motivation of [EL87] is the handling of fairness conditions in the context of model checking Kripke structures against CTL. The authors show NP-completeness for checking whether there exists a path in a Kripke structure satisfying an Emerson-Lei acceptance. Additionally, in [EL87] it is proven that CTL model checking with a disjunction of Streett formulas as fairness condition can be done in linear time in the size of the CTL formula, the size of the model and quadratic in the size of the fairness condition.

Results on model checking with different acceptance conditions does not only exist in the realm of Kripke structures, but in the realm of MDPs as well. Baier, Ciesinski and Größer [BGC09a] adapted the result of [EL87] for model checking Streett acceptance to MDPs. Chatterjee, Gaiser and Křetínský [CGK13] could improve significantly the speed of MDP analysis by the employment of deterministic automata with a generalized Rabin acceptance.

1.2 Contribution

This thesis offers new approaches for the analysis of Markov chains and MDPs that avoid the typical approach via a deterministic Rabin (or Streett) automaton. We always assume that the Markovian models are given in a discrete-time setting, and do not pursue a continues-time setting. For MDP analysis, the double-exponential time standard approach matches the lower bound, but from a practical point of view, it might still be possible to improve the efficiency by moving away from deterministic Rabin automata. We study restricted forms of non-determinism and a more extensive acceptance than Rabin acceptance.

For MDPs we turn to good-for-games automata and to Emerson-Lei acceptance. In this thesis we consider only deterministic Emerson-Lei automata and not combinations of the different approaches, e.g., GFG Emerson-Lei automata.

For Markov chains, the approach via a deterministic ω-automaton leaves a complexity gap to the PSPACE lower bound. We present a method that runs in polynomial time if a UBA specification is given. As LTL can be transformed into UBA with an exponential blow-up as upper bound, this algorithm runs in exponential time.

Good-for-games automata. In Chapter 3 we address the question whether good-for-games automata can be used for $1\frac{1}{2}$-player games. For this, we provide a new non-standard product construction with the goal of quantitative analysis of MDPs against ω-regular specifications represented by a good-for-games automaton. In the product we can resort to standard reachability analysis. As the process of building the product and the reachability analysis can be done in polynomial time, the overall approach is polynomial in time if a Markov decision process and a GFG automaton are given.

We adapt in a straightforward manner the proof of the double-exponential blow-up from LTL to deterministic automata [KV05] to the GFG setting. This double-exponential lower bound matches the double-exponential upper bound one would

obtain by combining the single-exponential translation from LTL to NBA and the single-exponential translation from NBA to GFG parity automata. Overall, the time complexity of our GFG-based method for MDP analysis matches the double-exponential lower bound if we are given an MDP and an LTL formula as input.

We also report on an implementation of the transformation of LTL to good-for-games parity automata described in [HP06] as well as their utility in probabilistic model checking.[2]

The contribution in Chapter 3 is based on [Kle+14].

Unambiguous Büchi automata. In Chapter 4 we explain the mistaken assumption of [BLW13b; BLW14] and provide an alternative polynomial time algorithm. The main part concentrates on the case where we have a uniform Markov chain, i.e., a Markov chain where at every position of every trace the symbols occur with the same probability. In this case the first step consists of an SCC analysis for every SCC in a bottom-up manner. This analysis decides whether the analyzed SCC consists of states inducing positive probability, and, if applicable, searches for a set of states within the SCC inducing probability 1. With the help of this state set, we can calculate the induced probability of every SCC state. Afterwards, the induced probability of states not contained in a positive SCC can be derived by solving a linear equation system.

The case of a uniform Markov chain and a UBA can be easily lifted to the general case of an arbitrary Markov chain.

The chapter concludes with several benchmarks. At first, we provide a benchmark on particular challenging UBA to compare the efficiency of two different positivity checks. For benchmarking the actual model checking process, we refer to the bounded retransmission protocol from the `PRISM` benchmark suite as a Markov chain model. Here, we evaluate pre-generated automata, and LTL formulas as property input. In case of pre-generated automata, we compare deterministic automata against unambiguous automata. In case of LTL, we compare the model checking approaches via deterministic automata, unambiguous automata and the automata-less method of [CY95]. As a last benchmark we compare the sizes of the generated automata for a selection of LTL formulas.

The contribution in Chapter 4 is based on [Bai+16].

Emerson-Lei acceptance. In Chapter 5 we consider the Emerson-Lei acceptance condition. We present a translation from two important LTL fragments into deterministic Emerson-Lei automata with an additional fallback to generic LTL-to-deterministic-ω-automata translators for full LTL. The two fragments are the safety-/cosafety fragment and a fragment which we call fairness fragment, and which subsumes the typical fairness conditions like unconditional, strong and weak fairness. Our translation features a product construction: We view an LTL formula as a positive Boolean combination of temporal formulas, i.e., formulas where the top-most

[2] We want to mention that the recently found problems in the value iteration (see [Bai+17]) did not affect any of our experiments in `PRISM`.

operator is a temporal operator. The temporal formulas are translated independently, and then combined via a product construction, where the acceptance reflects the Boolean structure of the input LTL formula. This product construction is enhanced by the knowledge about the subformulas, e.g., it is sufficient to check a fairness LTL formula after a cosafety LTL formula has already been satisfied if both formulas are combined via a conjunction.

As a second contribution in the field of Emerson-Lei acceptance we show how quantitative probabilistic model checking can be done for an MDP with the aid of DPLL-based techniques. It turns out that this DPLL-based algorithm mimics the standard behavior for checking a Rabin or Streett acceptance, and thus, works for both cases in polynomial time.

We conclude with a broad comparison between our newly developed tool `Delag` and state-of-the-art tools like `Rabinizer` or `SPOT`. This comparison takes place both by a direct automata comparison, and by the analysis of the WLAN handshaking protocol, an MDP model from the `PRISM` benchmark suite.

The contribution in Chapter 5 is based on [MS17] and not yet published material developed in a cooperation with Christel Baier, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, and Jan Strejček.

2 Preliminaries

In Chapter 2 we introduce the notations we use throughout the thesis.

2.1 Linear temporal logic

In this thesis we consider standard linear temporal logic (LTL) [Pnu77], a propositional logic augmented with the temporal operators $\mathcal{U}$ ("until") and $\bigcirc$ ("next").

Definition 2.1 (Syntax of LTL)**.** *A formula of LTL over a finite set of atomic propositions AP is given by the syntax:*

$$\varphi ::= \mathit{true} \mid a \mid \neg\varphi \mid \varphi \wedge \psi \mid \bigcirc\varphi \mid \varphi\,\mathcal{U}\,\psi \qquad \text{with } a \in Ap$$

We derive the usual abbreviations:

$$\begin{aligned}
\mathit{false} &= \neg\mathit{true} \\
\varphi \vee \psi &= \neg(\neg\varphi \wedge \neg\psi) \\
\varphi \rightarrow \psi &= \neg\varphi \vee \psi \\
\Diamond\varphi &= \mathit{true}\,\mathcal{U}\,\varphi && \text{"finally"} \\
\Box\varphi &= \neg\Diamond\neg\varphi && \text{"globally"} \\
\varphi\,\mathcal{R}\,\psi &= \neg\left(\neg\varphi\,\mathcal{U}\,\neg\psi\right) && \text{"release".}
\end{aligned}$$

We use $\mathrm{LTL}(M)$ for a set of temporal operators M to describe the fragment of LTL, where every temporal operator occurs in M. Additionally, $\mathrm{LTL}_{X,Y}(M) = \{X\varphi, Y\varphi : \varphi \in \mathrm{LTL}(M)\}$.

An ω-word w over the alphabet AP is an infinite sequence of sets of symbols $\sigma_0\,\sigma_1\,\sigma_2\ldots$. We denote the symbol at position i by $w[i] = \sigma_i$ and the infinite suffix $\sigma_i\,\sigma_{i+1}\ldots$ by $w\,[i\ldots]$.

Definition 2.2 (Semantics of LTL)**.** *The satisfaction relation $\models$ between an ω-word w and a formula φ is inductively defined as follows:*

$$\begin{aligned}
&w \models \mathit{true} \\
&w \models a \iff a \in w\,[0] \\
&w \models \neg\varphi \iff w \not\models \varphi \\
&w \models \bigcirc\varphi \iff w\,[1\ldots] \models \varphi \\
&w \models \varphi\,\mathcal{U}\,\psi \iff \exists i \geq 0.\,(w\,[i\ldots] \models \psi \wedge \forall j \in \{0,\ldots,i-1\}.w\,[j\ldots] \models \varphi)
\end{aligned}$$

For an LTL formula φ we define $\mathcal{L}(\varphi) = \{w \in (2^{AP})^\omega : w \models \varphi\}$ as the set of satisfying words. Two formulas φ, ψ are called equivalent, denoted by $\varphi \equiv \psi$, if $\mathcal{L}(\varphi) = \mathcal{L}(\psi)$.

The positive normal form demands that every occurrence of $\neg$ appears directly before an atomic proposition. An exhaustive rewriting with the following rewrite rules brings every LTL formula into positive normal form:

$$
\begin{array}{llll}
\neg \mathit{true} & \mapsto \mathit{false} & \neg \mathit{false} & \mapsto \mathit{true} \\
\neg(\varphi \wedge \psi) & \mapsto \neg\varphi \vee \neg\psi & \neg(\varphi \vee \psi) & \mapsto \neg\varphi \wedge \neg\psi \\
\neg \bigcirc \varphi & \mapsto \bigcirc \neg\varphi & \neg(\varphi \,\mathcal{U}\, \psi) & \mapsto \neg\varphi \,\mathcal{R}\, \neg\psi
\end{array}
\qquad
\begin{array}{ll}
\neg\neg\varphi & \mapsto \varphi \\
\varphi \rightarrow \psi & \mapsto \neg\varphi \vee \psi \\
\neg(\varphi \,\mathcal{R}\, \psi) & \mapsto \neg\varphi \,\mathcal{U}\, \neg\psi
\end{array}
$$

2.2 Automata over infinite words

ω-automata can be seen as language acceptors for infinite words. In this thesis we consider only ω-automata that exhibits non-deterministic branching. We do not allow universal branching as in alternating automata.

Definition 2.3. *An ω-automaton $\mathcal{A} = (Q, \Sigma, \delta, Q_0, \Phi)$ is a tuple, where*

- *Q is a non-empty, finite set of states,*
- *Σ is a finite alphabet,*
- *$\delta : Q \times \Sigma \to 2^Q$ is the (non-deterministic) transition function,*
- *$Q_0 \subseteq Q$ is the non-empty set of initial states and*
- *Φ is the acceptance condition.*

We denote by $\mathcal{A}[R]$ for $R \subseteq Q$ the automaton $\mathcal{A}$ with R as initial states, i.e., $\mathcal{A}[R] = (Q, \Sigma, \delta, R, \Phi)$. In case $R = \{q\}$ is a singleton, we omit the braces: $\mathcal{A}[q]$. We extend the transition function to $\delta : 2^Q \times \Sigma^* \to 2^Q$ in the standard way for subsets of Q and finite words over Σ. The size $|\mathcal{A}|$ of $\mathcal{A}$ denotes the number of states in $\mathcal{A}$. For complexity results we sometimes refer to the word length of $\mathcal{A}$ which is the length of the string when $\mathcal{A}$ is written in binary encoding on the tape of a Turing machine.

$\mathcal{A}$ is said to be *complete*, if $\delta(q, \sigma) \neq \varnothing$ for all states $q \in Q$ and all symbols $\sigma \in \Sigma$. $\mathcal{A}$ is called *deterministic*, if $|Q_0| = 1$ and $|\delta(q, \sigma)| \leq 1$ for all $q \in Q$ and $\sigma \in \Sigma$. Given states $q, p \in Q$ and a finite word $x = \sigma_1 \sigma_2 \dots \sigma_n \in \Sigma^*$ then a run for x from q to p is a sequence $q_0 q_1 \dots q_n \in Q^+$ with $q_0 = q$, $q_n = p$ and $q_{i+1} \in \delta(q_i, \sigma_{i+1})$ for $0 \leqslant i < n$. A *run* in $\mathcal{A}$ for an infinite word $w = \sigma_0 \sigma_1 \sigma_2 \dots \in \Sigma^\omega$ is a sequence $\rho = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \dots$ starting in an initial state q_0 such that $q_{i+1} \in \delta(q_i, \sigma_i)$ for all $i \in \mathbb{N}$. If the word w is clear, we sometimes omit the transitions and just write $\rho = q_0 q_1 \dots$.

We write $\inf(\rho)$ to denote the set of all states occurring infinitely often in ρ. A run ρ is called accepting, if it meets the acceptance condition Φ, denoted by $\rho \models \Phi$. As the syntactical description of the acceptance condition we use the syntax presented in [Bab+15], where the acceptance condition is denoted by a positive Boolean combination of $\mathrm{Fin}(Z)$ or $\mathrm{Inf}(Z)$ atoms with $Z \subseteq Q$ and with $\wedge$ and $\vee$ as allowed Boolean connectives. We call this acceptance an Emerson-Lei acceptance.

The semantics of $\mathrm{Fin}(Z)$ and $\mathrm{Inf}(Z)$ are defined in straight-forward manner: A run $\rho = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \dots$ is accepting for $\mathrm{Fin}(Z)$ if and only if $\inf(\rho) \cap Z = \varnothing$ holds, whereas ρ is accepting for $\mathrm{Inf}(Z)$ if and only if $\inf(\rho) \cap Z \neq \varnothing$ holds. $\mathrm{Fin}(\cdot)$ and $\mathrm{Inf}(\cdot)$ are dual to each other, i.e., every run ρ is accepting for $\mathrm{Inf}(Z)$ if and only if it is not accepting for $\mathrm{Fin}(Z)$, and analogously, ρ is accepting for $\mathrm{Fin}(Z)$ if and only if it is not accepting for $\mathrm{Inf}(Z)$. This dualism allows an easy complementation for deterministic Emerson-Lei automata by replacing every $\mathrm{Inf}(Z)$ with $\mathrm{Fin}(Z)$ and vice versa, and replacing every $\wedge$ with $\vee$ and vice versa as well.

We consider here the following six special types of acceptance conditions in particular and describe their constraints for infinite runs:

- *Büchi:* $\Phi = \mathrm{Inf}(Z)$ stands for a set of states, that needs to appear infinitely often.

- *generalized Büchi:* $\Phi = \bigwedge_i \mathrm{Inf}(Z_i)$ is a conjunction of Büchi acceptances, i.e., each Z_i has to appear infinitely often.

- *co-Büchi:* $\Phi = \mathrm{Fin}(Z)$ is the dual acceptance of Büchi acceptance.

- *parity:* Φ can be seen as a function $col : Q \to \mathbb{N}$ assigning to each state q a parity color and requiring that the least parity color appearing infinitely often is even.[1] As formal syntax we fix $\mathrm{Inf}(Z_0) \vee (\mathrm{Fin}(Z_1) \wedge (\mathrm{Inf}(Z_2) \vee (\mathrm{Fin}(Z_3) \wedge \dots)))$ with Z_i consisting of all states of color i.

- *generalized Rabin:* Φ is a disjunction of conjunctions, where every conjunction has at most one $\mathrm{Fin}(\cdot)$. Formally,

$$\Phi = \bigvee_{i \in \{1,\dots,n\}} \left(\mathrm{Fin}(U_i) \wedge \bigwedge_{j \in \{1,\dots,n_i\}} \mathrm{Inf}(L_{1,j}) \right),$$

  i.e., requiring that for one of the conjunctions the states in U_i appear at most finitely often while in $L_{i,j}$ for every $j \in \{1, \dots, n_i\}$ some state appears infinitely often. The term Rabin acceptance (without generalized) describes the special case where $n_i = 1$ for every i in $\{1, \dots, n\}$

[1] One can replace "least" by "maximal" and "even" by "odd" to get other versions of parity. No version of parity acceptance has more expressiveness, as one can change from "least" to "maximal" by reversing the colors, if they are seen as an ordered list. Analogously, for "even" and "odd" one can just add 1 to every color.

- *Streett:* Φ is dual to Rabin, i.e., it is a strong fairness condition. Syntactically, we fix $\bigwedge_{i\in\{1,\dots,n\}} \mathrm{Fin}(U_i) \vee \mathrm{Inf}(L_i)$ as Streett acceptance.

Büchi acceptance can be seen as a special case of parity acceptance which again can be seen as a special case of Rabin acceptance as well as Streett acceptance. We use the standard notations NBA (NPA, NRA, NSA) for non-deterministic Büchi (parity, Rabin, Streett) automata and DBA, DPA, DRA, DSA for their deterministic versions. In an analogous way, we define transition-based acceptance. Syntactically, transition-based acceptance uses atoms $\mathrm{Fin}(Z)$ and $\mathrm{Inf}(Z)$ with Z being a set of transitions, the rest transfers directly.

The language of $\mathcal{A}$, denoted by $\mathcal{L}_\omega(\mathcal{A})$, consists of all infinite words $w \in \Sigma^\omega$ that have at least one accepting run in $\mathcal{A}$, i.e., $w \in \mathcal{L}_\omega(\mathcal{A})$ if and only if there exists a run ρ for w with $\rho \models \Phi$. To simplify notifications, we write $\mathcal{L}_\omega(R)$ for $\mathcal{L}_\omega(\mathcal{A}[R])$ and $\mathcal{L}_\omega(q)$ for $\mathcal{L}_\omega(\mathcal{A}[q])$ if $\mathcal{A}$ is clear from the context.

It is well-known (see [Tho97; GTW02]) that the classes of languages recognizable by DRA, DSA or DPA, their non-deterministic version, and NBA are the same (the so-called ω-regular languages), while DBA are less powerful. For each LTL formula φ with atomic propositions in some finite set AP, the semantics of φ can be described as an ω-regular language $\mathcal{L}(\varphi)$ over the alphabet $\Sigma = 2^{AP}$ and there is an NBA $\mathcal{A}$ for φ (i.e., $\mathcal{L}(\varphi) = \mathcal{L}_\omega(\mathcal{A})$) whose size is at most exponential in the formula length $|\varphi|$ [WVS83; VW86].

There are several important subclasses of ω-regular languages, which we explain now. A safety language L is characterized by so-called bad prefixes, i.e., a set of finite words B such that $\mathrm{Pref}(L) \cap B = \varnothing$ where $\mathrm{Pref}(L) = \{u \in \Sigma^* : \exists w \in \Sigma^\omega . u\,w \in L\}$. A well-known (but incomplete) LTL fragment describing ω-regular safety languages is $\mathrm{LTL}(\mathcal{R}, \bigcirc)$. The dual of safety languages are cosafety languages, i.e., an ω-regular language L is cosafety, if $\Sigma^\omega \setminus L$ is a safety language. Analogously, every formula out of $\mathrm{LTL}(\mathcal{U}, \bigcirc)$ describes a cosafety language, but there are LTL formulas describing cosafety languages not in $\mathrm{LTL}(\mathcal{U}, \bigcirc)$.

As a third fragment we consider $\mathrm{LTL}_{\square\lozenge,\lozenge\square}(\lozenge, \square, \bigcirc)$ (and the Boolean combinations of it) which we call the fairness fragment. With this fragment one can enforce realistic behavior for example that certain transitions are taken infinitely often.

2.3 Automata over finite words

We use non-deterministic finite automata (NFA) as acceptors of regular languages over finite words. The syntax agrees with the syntax of ω-automata except that the acceptance condition is fixed to $\mathrm{Reach}(Z)$ with Z being a subset of Q. A *run* in a NFA $\mathcal{A}$ for a finite word $w = \sigma_0\,\sigma_1 \dots \sigma_n \in \Sigma^*$ is a sequence $\rho = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \dots \xrightarrow{\sigma_n} q_{n+1}$ starting in an initial state q_0 such that $q_{i+1} \in \delta(q_i, \sigma_i)$ for all $i \in \mathbb{N}$. The run $\rho = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \dots \xrightarrow{\sigma_n} q_{n+1}$ is accepting if and only if $q_{n+1} \in Z$. A word $\sigma_0 \dots \sigma_n$ is accepted, if and only if there exists an accepting run for the word. Analogously to infinite words, we write $\mathcal{L}_{\mathrm{fin}}(\mathcal{A})$ for the set of accepted words.

To obtain an equivalent deterministic finite automaton (DFA) $\mathcal{A}_{\text{det}}$ for an NFA $\mathcal{A} = (Q, \Sigma, \delta, Q_0, \text{Reach}\,(Z))$, we apply the powerset construction, also called Rabin-Scott construction [RS59]. The powerset tracks every possible run for a finite word in a sequence of powersets. More formally,

$$\mathcal{A}_{\text{det}} = (2^Q, \Sigma, \delta_{\text{det}}, \{Q_0\}, \text{Reach}\,(Z_{\text{det}})),$$

where

- $\delta_{\text{det}}(P, \sigma) = \left\{\bigcup_{q \in P} \delta(q, \sigma)\right\}$,
- $Z_{\text{det}} = \left\{P \in 2^Q : P \cap Z \neq \varnothing\right\}$.

For a simpler notation, we omit unnecessary brackets for singleton sets in the case of DFA, e.g., the above definition for $\delta_{\text{det}}(P, \sigma)$ simplifies to $\delta_{\text{det}}(P, \sigma) = \bigcup_{q \in P} \delta(q, \sigma)$.

2.4 Markovian models

Markovian models serve to describe probabilistic behavior. In this thesis, we restrict ourselves to Markov decision processes (MDP for short) as well as sometimes to a subclass of Markov decision processes, Markov chains. We only consider the discrete-time setting, i.e., the behavior evolves in discrete steps, in contrast to the continuous time-setting where the behavior occur in reference to a real-valued timeline.

Markov chains. For a clear and easy presentation we start with (discrete-time) Markov chains (DTMCs for short) and progress afterwards to (discrete-time) Markov decision processes.

Markov chains are an operational model for systems that exhibit solely probabilistic choices.

Definition 2.4. *A Markov chain is a tuple*

$$\mathcal{M} = (S, P, \iota, AP, \ell)$$

where

- S *is a finite set of states,*
- $P : S \times S \to [0, 1]$ *is the transition probability function satisfying:*

$$\sum_{s' \in S} P(s, s') \in \{0, 1\} \qquad \text{for all } s \in S,$$

- $\iota : S \to [0, 1]$ *is the initial distribution satisfying* $\sum_{s \in S} \iota(s) = 1$,
- $\ell : S \to 2^{AP}$ *is a labeling function.*

The size of a Markov chain, written as $|\mathcal{M}|$, is defined as its number of states. For complexity results we sometimes assume the Markov chain to be written in binary encoding on the tape of a Turing machine as input. We call the length of this input word length of the Markov chain.

Occasionally, we replace the initial distribution ι of a Markov chain $\mathcal{M}$ with the Dirac distribution $\mathsf{Dirac}[s]$ for a certain state s, where $\mathsf{Dirac}[s] : S \to [0,1]$ denotes the distribution mapping s to 1 and every other state to 0. We denote this Markov chain by $\mathcal{M}[s]$.

The last two components, AP and ℓ, serve to formalize properties of paths in $\mathcal{M}$. Formally, AP is a finite set of atomic propositions and $\ell : S \to 2^{AP}$ assigns to each state s the set $\ell(s)$ of atomic propositions that hold in s. Paths in $\mathcal{M}$ are finite or infinite sequences $\pi = s_0\ s_1\ s_2\ \ldots$ starting in the initial state s_0 that are built by consecutive steps, i.e., $P(s_i, s_{i+1}) > 0$ for all i. The trace of π is the word over the alphabet $\Sigma = 2^{AP}$ that arises by taking the projections to the state labels, i.e., $\mathit{trace}(\pi) = \ell(s_0)\,\ell(s_1)\,\ell(s_2)\ldots$.

Given a finite path $\hat{\pi} = s_0\, s_1\, \ldots\, s_n$ the cylinder set of $\hat{\pi}$, denoted $\mathit{Cyl}(\hat{\pi})$, is the set of infinite paths $\pi = s_0\, s_1\, \ldots$ such that $\hat{\pi} \in \mathrm{Pref}(\pi)$ (with $\mathrm{Pref}(\pi)$ as a short form for $\mathrm{Pref}(\{\pi\})$). The set of infinite paths is supposed to be equipped with the σ-algebra generated by the cylinder sets of the finite paths and the probability measure given by $\Pr\big(\, \mathit{Cyl}(s_0\, s_1\, \ldots\, s_n)\,\big) = \iota(s_0) \cdot \prod_{0 \le i < n} P(s_i, s_{i+1})$ where $a_1, \ldots, a_n \in \Sigma$. In notations like $\Pr_{\mathcal{M}}(\varphi)$ or $\Pr_{\mathcal{M}}(\mathcal{A})$ we identify LTL formulas φ and ω-automata $\mathcal{A}$ with their languages. For the mathematical details of the underlying σ-algebra and probability measure, we refer to [Put94; BK08].

Occasionally, we also consider Markov chains with transition labels in some alphabet Σ. These are defined as triples $\mathcal{M} = (S, P, \iota)$ where S and ι are as above and the transition probability function is of the type $P : S \times \Sigma \times S \to [0,1]$ such that $\sum_{(a,s') \in \Sigma \times S} P(s, a, s') = 1$ for all states $s \in S$. If $L \subseteq \Sigma^\omega$ is measurable, then $\Pr_{\mathcal{M}}(L)$ denotes the probability measure of the set of infinite paths π where the projection to the transition labels constitutes a word in L. Furthermore, if $\mathcal{M}[\Sigma] = (S, P, \iota)$ is a transition-labeled Markov chain where $S = \{s\}$ is a singleton and $P(s, a, s) = 1/|\Sigma|$ for all symbols $a \in \Sigma$, then $\Pr_{\mathcal{M}[\Sigma]}(L) = \Pr(L)$ for all measurable languages L.

We refer to the positivity problem and almost universality problem every now and then. The positivity problem asks whether a positive probability given an ω-regular language L holds, i.e., whether $\Pr_{\mathcal{M}}(L) > 0$ holds. Complementary, the almost universality problem asks whether $\Pr_{\mathcal{M}}(L) = 1$ holds.

The terms positivity and almost-surely directly transfers to automata if we assume a uniform Markov chain, i.e., a Markov chain whose traces have at every position every symbol with the same probability.

Markov decision processes. MDPs are an operational model for systems that exhibit both non-deterministic and probabilistic choices.

Definition 2.5. *A Markov decision process is a tuple*

$$\mathcal{M} = (S, \mathit{Act}, P, \iota, AP, \ell)$$

where

- S *is a finite set of states,*
- Act *is a finite set of actions,*
- $P : S \times Act \times S \to [0,1]$ *is the transition probability function satisfying:*

$$\sum_{s' \in S} P(s, \alpha, s') \in \{0, 1\} \qquad \text{for all } s \in S,\ \alpha \in Act,$$

- $\iota : S \to [0,1]$ *is the initial distribution satisfying* $\sum_{s \in S} \iota(s) = 1$, *and*
- $\ell : S \to 2^{AP}$ *is a labeling function.*

The definitions of the size and of the word length of an MDP transfers directly. Analogously, the notions $\mathcal{M}[\mu]$ and $\mathcal{M}[s]$ for a distribution μ and a state s are the same as for Markov chains.

We write $Act(s)$ for the set of actions α that are enabled in s, i.e., $P(s, \alpha, s') > 0$ for some $s' \in S$, in which case $s' \mapsto P(s, \alpha, s')$ is a distribution formalizing the probabilistic effect of taking action α in state s. We refer to the triples (s, α, s') with $P(s, \alpha, s') > 0$ as a step. The choice between the enabled actions is viewed to be non-deterministic. For technical reasons, we require $Act(s) \neq \varnothing$ for all states s. The last two components, AP and ℓ, serve to formalize properties of paths in $\mathcal{M}$. Formally, AP is a finite set of atomic propositions and $\ell : S \to 2^{AP}$ assigns to each state s the set $\ell(s)$ of atomic propositions that hold in s. Paths in $\mathcal{M}$ are finite or infinite sequences $\pi = s_0\, \alpha_0\, s_1\, \alpha_1\, s_2\, \alpha_2 \ldots$ starting in the initial state s_0 that are built by consecutive steps, i.e., $P(s_i, \alpha_i, s_{i+1}) > 0$ for all i. The trace of π is the word over the alphabet $\Sigma = 2^{AP}$ that arises by taking the projections to the state labels, i.e., $trace(\pi) = \ell(s_0)\, \ell(s_1)\, \ell(s_2) \ldots$. For an LTL formula φ over AP we write $\pi \models \varphi$ if $trace(\pi) \in \mathcal{L}(\varphi)$.

MDPs can be seen as stochastic games, also called a $1\frac{1}{2}$-player games. The first (full) player resolves the non-deterministic choice by selecting an enabled action α of the current state s. The second (half) player behaves probabilistically and selects a successor state s' with $P(s, \alpha, s') > 0$. Strategies for the full player are called *schedulers*. In general, they can be history-dependent, i.e., a scheduler is a function $\mathfrak{s} \colon (S \times Act)^* \times S \to Act$ selecting the next action given the current path prefix. We call a path $\pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots$ an $\mathfrak{s}$-path if $\alpha_i = \mathfrak{s}(s_0, \alpha_0, s_1, \alpha_1, \ldots, \alpha_{i-1}, s_i)$ for all $i \geq 0$.

Since the behavior of $\mathcal{M}$ is purely probabilistic if some scheduler $\mathfrak{s}$ is fixed, one can reason about the probability of path events. One can construct a (possibly infinite) Markov chain $\mathcal{M}_\mathfrak{s}$ induced by $\mathcal{M}$ and $\mathfrak{s}$: $\mathcal{M}_\mathfrak{s} = (S_\mathfrak{s}, P_\mathfrak{s}, \iota, AP, \ell_\mathfrak{s})$ where

- $S_\mathfrak{s}$ are all finite, non-empty sequences of state-action pairs finished by a state in $(S \times Act)^* S$,

- If $\mathfrak{s}(s_0 \xrightarrow{\alpha_0} \dots \xrightarrow{\alpha_{n-1}} s_n) = \alpha_n$:
  $P_{\mathfrak{s}}(s_0 \xrightarrow{\alpha_0} \dots \xrightarrow{\alpha_{n-1}} s_n, s_0 \xrightarrow{\alpha_0} \dots \xrightarrow{\alpha_{n-1}} s_n \xrightarrow{\alpha_n} s_{n+1}) = P(s_n, \alpha_n, s_{n+1})$,
- If $\mathfrak{s}(s_0 \xrightarrow{\alpha_0} \dots \xrightarrow{\alpha_{n-1}} s_n) \neq \alpha_n$:
  $P_{\mathfrak{s}}(s_0 \xrightarrow{\alpha_0} \dots \xrightarrow{\alpha_{n-1}} s_n, s_0 \xrightarrow{\alpha_0} \dots \xrightarrow{\alpha_{n-1}} s_n \xrightarrow{\alpha_n} s_{n+1}) = 0$, and
- $\ell_{\mathfrak{s}}(s_0 \xrightarrow{\alpha_0} \dots \xrightarrow{\alpha_{n-1}} s_n) = \ell(s_n)$.

If L is an ω-regular language, then $\Pr^{\mathfrak{s}}_{\mathcal{M}}(L)$ denotes the probability under $\mathfrak{s}$ for the set of infinite paths π with $trace(\pi) \in L$ which equals $\Pr_{\mathcal{M}_{\mathfrak{s}}}(L)$.

If $\mathcal{M}$ is clear from the context, we omit $\mathcal{M}$ in notations like $\Pr^{\mathfrak{s}}_{\mathcal{M}}$ and just write $\Pr^{\mathfrak{s}}$. Analogously, $\Pr^{\mathfrak{s}}_{s}$ for $\Pr^{\mathfrak{s}}_{\mathcal{M}[s]}$.

For a worst-case analysis of a system modeled by an MDP $\mathcal{M}$, one ranges over all schedulers (i.e., all possible resolutions of the non-determinism) and considers the maximal or minimal probabilities for some ω-regular language L. Depending on whether L represents a desired or undesired path property, the quantitative worst-case analysis amounts to computing $\Pr^{\min}_{\mathcal{M}}(\varphi) = \min_{\mathfrak{s}} \Pr^{\mathfrak{s}}_{\mathcal{M}}(L)$ or $\Pr^{\max}_{\mathcal{M}}(L) = \max_{\mathfrak{s}} \Pr^{\mathfrak{s}}_{\mathcal{M}}(L)$. The existence of such schedulers is well-known, see, e.g., [Put94; FV96].

We introduced only deterministic schedulers, i.e., schedulers that choose a particular action given a history. There are also randomized schedulers, that choose a distribution over the actions for a finite history. As the probability $\Pr^{\max}_{\mathcal{M}}(L)$ for an ω-regular language is independent whether one ranges over all randomized schedulers or only over all deterministic schedulers, and analogously for $\Pr^{\min}_{\mathcal{M}}(L)$, it is sufficient for our purposes to consider only deterministic schedulers. For further informations on randomized schedulers we refer to, e.g., [Put94].

Markov chains are a special case of MDPs, where $|Act(s)| = 1$ for every state $s \in S$. This results in the existence of exactly one scheduler, and therefore $\Pr^{\min}_{\mathcal{M}}$ and $\Pr^{\max}_{\mathcal{M}}$ coincides.

Analogously to Markov chains, we refer to the positivity problem and almost universality problem every now and then. In the positivity problem we ask whether a positive probability given an ω-regular language L holds for at least one scheduler, i.e., whether $\Pr^{\max}_{\mathcal{M}}(L) > 0$ holds. Complementary, in the almost universality problem we ask whether $\Pr^{\min}_{\mathcal{M}}(L) = 1$ holds.

2.4.1 Analysis of Markovian models under LTL specifications

The standard automata-based analysis of Markovian model (we assume we are given an MDP or a Markov chain) relies on a product construction of the Markovian model with a deterministic automaton, where the automaton serves as a monitor signaling accepted behavior. The standard approach is agnostic to the method transforming for LTL to a deterministic ω-automaton. Therefore several methods have been developed, the two main directions being a direct translation or taking an intermediate step via non-deterministic Büchi automata.

However, there is an unavoidable double-exponential blow-up in the transformation from LTL to deterministic ω-automata, see [AL04; KV05; KR11]. This raises the

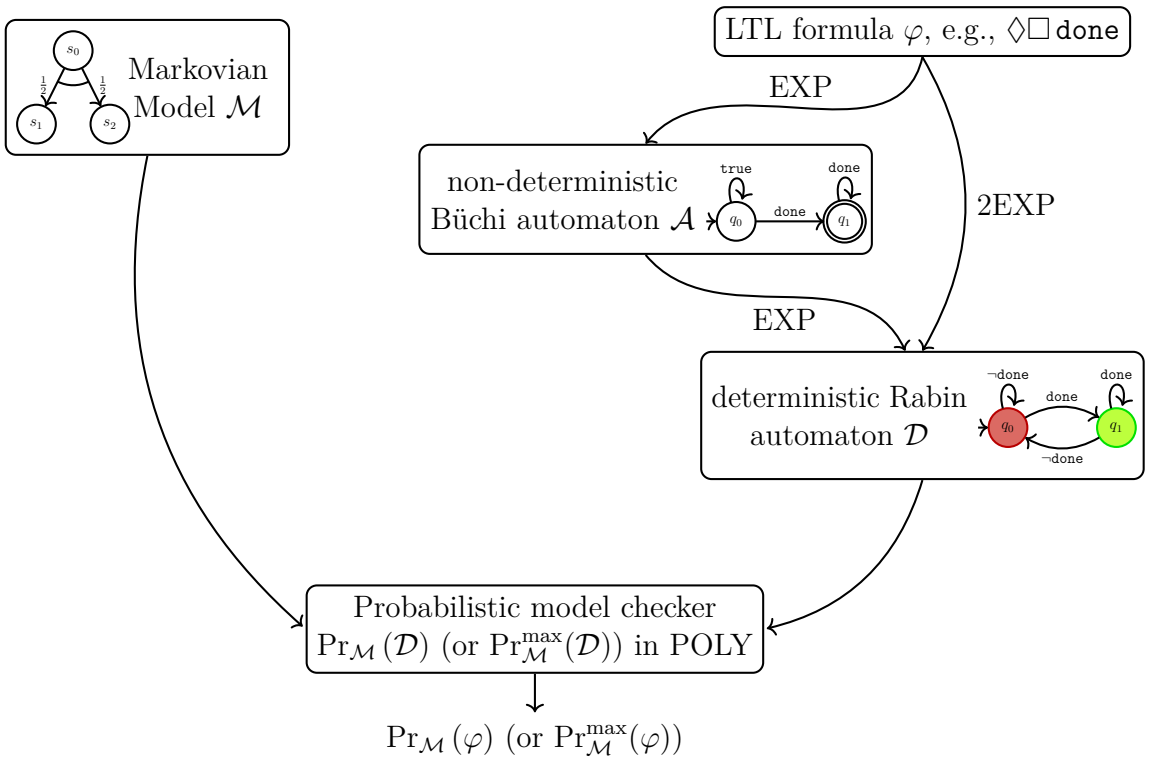


Figure 2.1: General scheme for probabilistic model checking with LTL as implemented in PRISM. The edge labelings EXP and 2EXP are upper and lower bounds for an exponential (respectively double-exponential) blow-up.

question whether non-deterministic Büchi automata can be used directly for the analysis of Markovian models, since LTL can be translated into non-deterministic Büchi automata with a single-exponential blow-up. This question has to be declined, as the positivity and the almost universality problem for strongly connected NBA are PSPACE-complete, see [Var85; CY95] and also Theorem 4.43.

Formally, the product of a Markov decision process $\mathcal{M} = (S, Act, P, \iota, AP, \ell)$ and a complete deterministic ω-automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, \Phi)$ is defined as the Markov decision process

$$\mathcal{M} \otimes \mathcal{A} = (S \times Q, Act, P', \iota', \{\tau\})$$

where

- $P'(\langle s, q\rangle, \alpha, \langle s', q'\rangle) = \begin{cases} P(s, \alpha, s') & \text{if } q' = \delta(q, \ell(s')) \\ 0 & \text{otherwise} \end{cases}$
- $\iota'(s, q) = \begin{cases} \iota(s) & \text{if } q = \delta(q_0, \ell(s)) \\ 0 & \text{otherwise} \end{cases}$

The product should be seen as the original Markov decision process annotated with the states of $\mathcal{A}$, indicating whether a certain behavior is accepted or not.

As Markov chains are essentially Markov decision processes with exactly one enabled action per state, the product definition carries over directly.

As a second step we need to analyse the so-called end-components, since almost all paths will end in an end-component and visit every state in it infinitely often [Alf97]. An *end-component* is a pair (T, A) where

- $T \subseteq S$ is a non-empty subset of states,
- $A(s) \subseteq Act(s)$ is a non-empty set of enabled actions for every $s \in T$,
- for every $s \in T$ and $\alpha \in A(s)$ $\mathrm{Post}\,(s, \alpha) = \{t \in S : P(s, \alpha, t) > 0\} \subseteq T$, and
- (T, A) induces a strongly connected component.

We call an end-component (T, A) maximal (MEC for short), if there is no end-component $(T', A') \neq (T, A)$ with $T \subseteq T'$ and $A(s) \subseteq A'(s)$ for all $s \in T$.

For an infinite path $\pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} \ldots$ we describe its limit, denoted by $\mathrm{Limit}(\pi)$, as the pair (T, A) with T being the set of infinitely often visited states and $A : T \to 2^{Act}$ being the function mapping every state to its set of infinitely often taken actions.

Lemma 2.6 (see Theorem 3.2 of [Alf97])**.** *Let $\mathcal{M}$ be an MDP, s a state in $\mathcal{M}$, and $\mathfrak{s}$ a scheduler for $\mathcal{M}$. Then*

$$\Pr^{\mathfrak{s}}_{\mathcal{M}[s]}\left(\pi \in Paths(s) : \mathrm{Limit}(\pi) \text{ is an end-component}\right) = 1$$

In the special case of a Markov chain, the end-components degenerate to bottom strongly connected components (BSCCs for short), i.e., strongly connected components without outgoing transitions. Thus, Lemma 2.6 can be reformulated into

Lemma 2.7 (see Corollary 3.1 of [Alf97])**.** *Let $\mathcal{M}$ be a Markov chain, and s a state in $\mathcal{M}$. Then*

$$\Pr_{\mathcal{M}[s]}(\pi \in Paths(s) : \inf(\pi) \text{ is a BSCC}) = 1$$

With Lemma 2.6 we can now analyse every end-component whether it is accepting or not. In this section we restrict ourselves to the Rabin acceptance. For a deeper inspection of the MEC analysis under the Emerson-Lei acceptance, we refer to Section 5.3.

Let $\bigvee_{i \in \{1,\ldots,n\}} (\mathrm{Fin}\,(E_i) \wedge \mathrm{Inf}\,(F_i))$ be a Rabin acceptance and (T, A) a MEC. Then, we call (T, A) accepting if and only if it contains an end-component (T', A') such that there is a Rabin pair $\mathrm{Fin}\,(E_i) \wedge \mathrm{Inf}\,(F_i)$ with $T' \cap E_i = \varnothing$ and $T' \cap F_i \neq \varnothing$. We call the set of states being in an accepting maximal end-component U. With the knowledge of the accepting end-components, we can reduce MDP analysis to calculation the probability of reaching a state in U.

Theorem 2.8. *Let $\mathcal{M}$ be an MDP, φ an LTL formula, $\mathcal{A}$ be a DRA equivalent to φ, and U the set of all states in an accepting end-component in $\mathcal{M} \otimes \mathcal{A}$. Then*

$$\Pr^{\max}_{\mathcal{M}}(\varphi) = \Pr^{\max}_{\mathcal{M} \otimes \mathcal{A}}(\Diamond U)$$

$$
\begin{aligned}
x_s &= 1 && \text{for all } s \in U \\
x_s &= 0 && \text{for all } s \text{ such that } U \text{ is not reachable from } s \\
0 \leq x_s &\leq 1 && \text{for all } s \text{ such that } U \text{ is reachable from } s \text{ and } s \notin U \\
x_s &\geq \sum_{t \in S} P(s, \alpha, t) \cdot x_t && \text{for all } \alpha \in Act \text{ and } s \text{ such that } U \text{ is reachable from } s \text{ and } s \notin U
\end{aligned}
$$

and minimize $\sum_{s \in S} x_s$

Figure 2.2: Linear program for calculating $(x_s)_{s \in S}$ with $x_s = \Pr^{\max}_{\mathcal{M}[s]}(\Diamond U)$ for an MDP $\mathcal{M}$ and state set U.

$$
\begin{aligned}
x_s &= 1 && \text{for all } s \in U \\
x_s &= 0 && \text{for all } s \text{ such that } U \text{ is not reachable from } s \\
x_s &= \sum_{t \in S} P(s, t) \cdot x_t && \text{for all } s \text{ such that } U \text{ is reachable from } s \text{ and } s \notin U
\end{aligned}
$$

Figure 2.3: Linear equation system for calculating $(x_s)_{s \in S}$ with $x_s = \Pr_{\mathcal{M}[s]}(\Diamond U)$ for a Markov chain $\mathcal{M}$ and state set U.

Now we highlight how to calculate the maximal probability to satisfy a reachability property:

Theorem 2.9 (see Theorem 3.5 of [Alf97])**.** *Let $\mathcal{M}$ be an MDP with S being the states and $U \subseteq S$. Then the unique solution $(x_s)_{s \in S}$ of the linear program in Figure 2.2 agrees with $\Pr^{\max}_{\mathcal{M}[s]}(\Diamond U)$.*

In case of a Markov chain the linear program of Theorem 2.9 can be replaced by the system of linear equations in Figure 2.3.

As the last step, we sum over the values of the initial distribution for every state s:

$$\Pr^{\max}_{\mathcal{M}}(\Diamond U) = \sum_{s \in S} \iota(s) \cdot \Pr^{\max}_{\mathcal{M}[s]}(\Diamond U)$$

Given a deterministic ω-automaton $\mathcal{D}$ all necessary steps for calculating $\Pr^{\max}_{\mathcal{M}}(\mathcal{D})$ can be done in polynomial time in the word length of $\mathcal{M}$ and $\mathcal{D}$. The translation from LTL to deterministic ω-automata requires double-exponential time in the worst-case, so for the overall complexity we obtain:

Theorem 2.10 (see [Var85])**.** *Deciding $\Pr^{\max}_{\mathcal{M}}(\varphi) > 0$ for an MDP $\mathcal{M}$ and an LTL formula φ is in 2EXPTIME.*

Despite Theorem 2.10 is formulated for qualitative analysis, the quantitative analysis can be carried out also in 2EXPTIME with the means as explained above.

The matching lower bound has been proven by Courcoubetis and Yannakakis via a reduction from the membership problem for exponential space bounded, alternating Turing machines:

Theorem 2.11 (see Theorem 3.2.1. of [CY95])**.** *Deciding* $\Pr^{\max}_{\mathcal{M}}(\varphi) > 0$ *for an MDP* $\mathcal{M}$ *and an LTL formula* φ *is 2EXPTIME-complete.*

The lower bound of Theorem 2.11 changes to a PSPACE lower bound in the case of Markov chains:

Theorem 2.12 (see [Var85])**.** *Deciding* $\Pr_{\mathcal{M}}(\varphi) > 0$ *for a Markov chain* $\mathcal{M}$ *and an LTL formula* φ *is PSPACE-hard.*

Despite there are several algorithms with a better worst-case time-complexity than 2EXPTIME (see Section 1.1.2), the usage of deterministic automata for the Markov chain analysis under LTL can be seen as standard, as it is the way `PRISM` and `STORM` deals with Markov chains and LTL.

3 Good-for-games Automata

A desire to avoid deterministic ω-automata occurred not only in the area of probabilistic model checking, but in the area of synthesis of reactive systems as well [KV05; KPV06; PPS06; SF07]. In 2006 Henzinger and Piterman [HP06] proposed the so-called good-for-games property for non-deterministic automata, a restricted form of non-determinism. This property has been independently proposed by Colcombet [Col09] for weighted automata, but here it is called history-deterministic. Henzinger and Piterman also developed an algorithm, which we call HP-algorithm, for constructing a good-for-games parity automaton out of an NBA, aimed at a compact symbolic representation. Very recently, at STACS 2018, Kuperberg and Majumdar [KM18] presented a modified breakpoint construction for transforming non-deterministic co-Büchi automata to good-for-games automata and sketched a generalisation to non-deterministic Büchi automata.

In a good-for-games automaton, the non-determinism can be resolved in an incremental way for every accepted word without look-ahead. The formal definition of GFG ω-automata [HP06] relies on a game-based view of ω-automata. Given a complete ω-automaton $\mathcal{A}$ as before, we consider $\mathcal{A}$ as the game arena of an infinite, turn-based 2-player game, called *monitor game*: if the current state is q, then player 1 chooses a symbol $\sigma \in \Sigma$ whereas the other player (player 0) has to answer by a successor state $q' \in \delta(q, \sigma)$, i.e., resolves the non-determinism. In the next round q' becomes the current state. A *play* is a maximal alternating sequence $\varsigma = q_0\,\sigma_0\,q_1\,\sigma_1\,q_2\,\sigma_2\,\ldots$ of states and (action) symbols in the alphabet Σ starting with an initial state q_0. Intuitively, the σ_i's are the symbols chosen by player 1 and the q_i's are the states chosen by player 0 in round i. Player 0 wins the play ς if whenever $\varsigma|_\Sigma = \sigma_0\,\sigma_1\,\sigma_2\ldots \in \mathcal{L}_\omega(\mathcal{A})$ then $\varsigma|_Q = q_0\,q_1\,q_2\ldots$ is an accepting run. A strategy for player 0 is a function $\mathfrak{f} : (Q \times \Sigma)^* \to Q$ with $\mathfrak{f}(\ldots\,q\,\sigma) \in \delta(q, \sigma)$ and $\mathfrak{f}(\varepsilon) \in Q_0$. A play $\varsigma = q_0\,\sigma_0\,q_1\,\sigma_1\,q_2\ldots$ is said to be $\mathfrak{f}$-*conform* or an $\mathfrak{f}$-play if $q_i = \mathfrak{f}(q_0\,\sigma_0\,\ldots\,\sigma_{i-2}\,q_{i-1}\sigma_{i-1})$ for all $i \geq 1$. An automaton $\mathcal{A}$ is called *good-for-games* if there is a strategy $\mathfrak{f}$ such that player 0 wins each $\mathfrak{f}$-play. Such strategies will be called *GFG-strategies* for $\mathcal{A}$. Obviously, each complete deterministic automaton enjoys the GFG property.

Example 3.1. *The first example, depicted in Figure 3.1, is a good-for-games Büchi automaton. It accepts the language $\mathcal{L}(\Box\Diamond a)$. This automaton is non-deterministic for the symbols that contain a, allowing the choice of either staying in the current state or switching to the other. A GFG strategy for player* 0 *is the following strategy $\mathfrak{f}$: If the current symbol contains the atomic proposition a, then player* 0 *chooses to go to state q_1, otherwise the strategy advises going to state q_0.*

This strategy ensures that all words accepted by $\mathcal{L}_\omega(\mathcal{A})$ are indeed accepted, as the

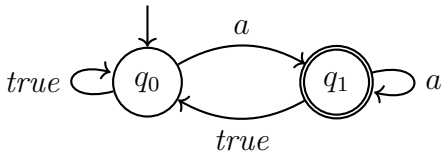


Figure 3.1: Good-for-games NBA $\mathcal{A}$ for $\square\lozenge a$. Accepting Büchi states are marked with a double circle.

only way for player 1 *to generate a word* $w \in \mathcal{L}_\omega(\mathcal{A})$ *is to choose infinitely many symbols with* a*. But then player* 0 *visits infinitely often the accepting state* q_1*, since every time a symbol with* a *is selected, player* 0 *moves to* q_1 *via his strategy. So for an accepted word the* $\mathfrak{f}$*-conform play contains infinitely many accepting states* q_1 *and thus the projection to the automata states delivers an accepting run.*

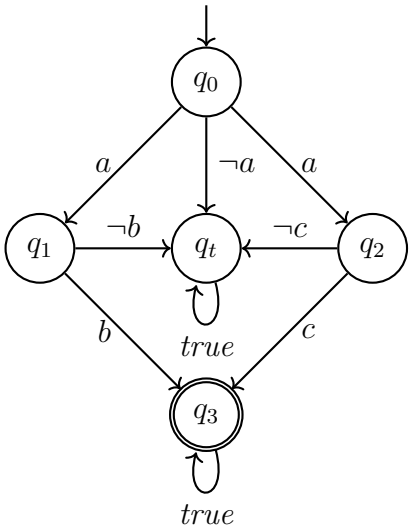


Figure 3.2: NBA $\mathcal{B}$ for $a \wedge \bigcirc(b \vee c)$, not good-for-games.

Example 3.2. *The second NBA, shown in Figure 3.2, is not good-for-games. Let player* 1 *pick the symbol* $\{a\}$ *in the beginning. Now there exists a non-deterministic choice between* q_1 *and* q_2*. Assume that the strategy for player* 0 *moves to* q_1*. Then, player* 1 *can choose the symbol* $\{c\}$*, leading to the trap state* q_t *from which no accepting run is possible. However, every word starting with* $\{a\}\{c\}$ *is an accepted word and thus choosing* q_1 *in the first choice is not a valid move for a GFG strategy. In the other case, if the strategy of player* 0 *moves to* q_2 *after the first symbol* $\{a\}$*, player* 1 *can choose the symbol* $\{b\}$*, trapping the run in the non-accepting* q_t *again. So, for every accepted word* $\{a\}\{b\}\ldots$ *the conform play again does not yield an accepted run and thus choosing* q_2 *is not a valid move for a GFG strategy either.*

We obtain that there does not exist a GFG strategy, because in the initial state q_0 *player* 0 *would need to know which of the two symbols player* 1 *will choose in the*

second step. So player 0 *would need the ability to look-ahead, which is impossible for GFG automata.*

The automaton in Figure 3.1 is *determinizable-by-pruning (DBP)*. Determinizable-by-pruning [Bok+13] means that an equivalent deterministic automaton is embedded, i.e., one can remove certain states and transitions, and obtain an equivalent deterministic automaton. [Bok+13] demonstrates the existence of GFG automata that are not determinizable-by-pruning. Colcombet studied GFG automata on finite words in more depth and discovered, that every GFG NFA is determinizable-by-pruning [Col12]. The influence of different acceptance conditions for good-for-games automata has been covered in [KS15; BKS17]. The authors of [KS15] have proven, that every GFG Büchi automaton can be determinized to a DBA with a quadratic blow-up in the state space at most. Thus, GFG Büchi automata are as expressive as DBA. The situation differs for GFG co-Büchi automata, where an exponential blow-up may be unavoidable, but since every GFG co-Büchi automaton is also a non-deterministic co-Büchi automaton, and non-deterministic co-Büchi automata have the same expressiveness as deterministic co-Büchi automata, GFG co-Büchi automata and deterministic co-Büchi automata are equivalent concerning expressiveness. For standard acceptance conditions like Rabin, Streett, Emerson-Lei, or parity, all GFG automata cover ω-regularity, since deterministic automata are trivially good-for-games. [BKS17] consider so-called *typeness* for (tight) good-for-games automata. Tightness enforces that a good-for-games automaton does not contain any redundant states or transitions that are not used by any GFG strategy. For explaining *typeness*, we assume a good-for-games Rabin automaton $\mathcal{A}$, whose language can be recognized by a good-for-games Büchi automaton. Then, there exists an equivalent good-for-games Büchi automaton on the same graph structure as $\mathcal{A}$. So, tight good-for-games Rabin automata are *Büchi*-type. The same notion transfers to the co-Büchi condition: For every tight GFG Streett automaton $\mathcal{A}$, that is co-Büchi-realizable, there is an equivalent GFG co-Büchi automaton on the same graph structure as $\mathcal{A}$.

Contribution. We present a new approach for probabilistic model checking based on good-for-games automata, and show how to compute maximal or minimal probabilities for path properties in MDPs. If we assume that the path properties are specified by a GFG automaton, we achieve polynomial time complexity in both the word length of the given MDP and GFG automaton if the GFG automaton has one of the standard acceptances (Büchi, parity, Rabin, Streett). If the path properties are specified by an LTL formula, we achieve a time complexity polynomial in the size of the given MDP and double-exponential in the size of the LTL formula.

We evaluate this GFG-based approach empirically using our symbolic implementation of the HP-algorithm, including several variants, using binary decision diagrams. We compare the performance of the HP-algorithm with the standard determinization approach of Safra, relying on the implementation `ltl2dstar` [KB06; KB07]. To compare the performance in actual probabilistic model checking, we have extended the probabilistic model checker `PRISM` [KNP04] and evaluate the GFG-based approach on

the IEEE802.11 handshaking protocol as well as on the dining philosopher's problem.

3.1 Automata-based Analysis of Markov Decision Processes

We address the task to compute the maximal or minimal probability in an MDP $\mathcal{M}$ for the path property imposed by a non-deterministic ω-automaton $\mathcal{A}$. The standard approach, see, e.g., [BK08], assumes $\mathcal{A}$ to be deterministic and relies on a product construction where states in $\mathcal{M}$ are augmented by the current state of $\mathcal{A}$. Thus, $\mathcal{M} \otimes \mathcal{A}$ can be seen as a refinement of $\mathcal{M}$ since $\mathcal{A}$ does not affect $\mathcal{M}$'s behaviors, but attaches information on $\mathcal{A}$'s current state for the prefixes of the traces induced by the paths of $\mathcal{M}$.

In the context of MDP analysis, we assume w.l.o.g. that a good-for-games automaton has exactly one initial state. For a good-for-games automaton with several initial states, we pick an arbitrary GFG strategy $\mathfrak{f}$, and set the unique initial state q_0 to the state $\mathfrak{f}(\varepsilon)$ chosen by $\mathfrak{f}$ for the empty word.

We now modify the standard definition of the product of an MDP with a non-deterministic ω-automata (with a unique initial state). The crucial difference is that the actions are now pairs $\langle \alpha, p \rangle$ consisting of an action in $\mathcal{M}$ and a state in $\mathcal{A}$, representing the non-deterministic alternatives in both the MDP $\mathcal{M}$ and the automaton $\mathcal{A}$. Formally, let $\mathcal{M} = (S, Act, P, \iota, AP, \ell)$ be an MDP and $\mathcal{A} = (Q, \Sigma, \delta, q_0, \Phi)$ a complete non-deterministic ω-automaton with $\Sigma = 2^{AP}$. The product MDP is

$$\mathcal{M} \otimes \mathcal{A} \;=\; (S \times Q, Act \times Q, P', \iota', Q, \ell')$$

where the transition probability function P' is given by $P'(\langle s, q \rangle, \langle \alpha, p \rangle, \langle s', q' \rangle) = P(s, \alpha, s')$ if $p = q' \in \delta(q, \ell(s))$. In all other cases $P'(\langle s, q \rangle, \langle \alpha, p \rangle, \langle s', q' \rangle) = 0$. The initial distribution is given by $\iota'(\langle s, q \rangle) = \iota(s)$ if $q = q_0$ and $\iota'(\langle s, q \rangle) = 0$ in all other cases. The assumption that $\mathcal{A}$ is complete yields that for each $\alpha \in Act(s)$ there is some action $\langle \alpha, q' \rangle \in Act(\langle s, q \rangle)$ for all states s in $\mathcal{M}$ and q in $\mathcal{A}$. In the product, the states of the automaton serve as the atomic propositions and the labeling function is given by $\ell'(\langle s, q \rangle) = \{q\}$, i.e., simply lifting the automaton state to the product. This allows us to consider the traces in $\mathcal{M} \otimes \mathcal{A}$ simply as words over the alphabet Q. Likewise, $\mathcal{A}$'s acceptance condition Φ can be seen as a language over Q, which permits treating Φ as a property that the paths in $\mathcal{M} \otimes \mathcal{A}$ might or might not have. In particular, for $\Pr^{\max}_{\mathcal{M} \otimes \mathcal{A}}(\Phi)$, Φ corresponds to the set of paths in the product where the projection on the $\mathcal{A}$-states yields an accepting path in $\mathcal{A}$.

Theorem 3.3. *For each MDP $\mathcal{M}$ and non-deterministic ω-automaton $\mathcal{A}$ as above:*

(a) $\Pr^{\max}_{\mathcal{M} \otimes \mathcal{A}}(\Phi) \;\leq\; \Pr^{\max}_{\mathcal{M}}\big(\mathcal{L}_\omega(\mathcal{A})\big)$

(b) *If $\mathcal{A}$ is good-for-games, then:* $\Pr^{\max}_{\mathcal{M} \otimes \mathcal{A}}(\Phi) \;=\; \Pr^{\max}_{\mathcal{M}}\big(\mathcal{L}_\omega(\mathcal{A})\big)$

Proof. We first observe that by the definition of the transition probability function P' we have:

- If $\pi' = \langle s_0, q_0\rangle\, \gamma_0\, \langle s_1, q_1\rangle\, \gamma_1\, \langle s_2, q_2\rangle\, \gamma_2 \ldots$ is a path in $\mathcal{M} \otimes \mathcal{A}$ where $\gamma_i = \langle \alpha_i, p_i\rangle$, then $p_i = q_{i+1}$ and $\pi'|_{\mathcal{M}} = s_0\, \alpha_0\, s_1\, \alpha_1\, s_2\, \alpha_2 \ldots$ is a path in $\mathcal{M}$ and $\pi'|_{\mathcal{A}} = q_0\, q_1\, q_2 \ldots$ is a run in $\mathcal{A}$ for the word

$$\mathit{trace}\big(\,\pi'|_{\mathcal{M}}\,\big) \;=\; \ell(s_0)\,\ell(s_1)\,\ell(s_2)\ldots \in \big(2^{AP}\big)^{\omega}$$

  In this case, we have:

$$\pi' \models \Phi \quad \text{iff} \quad \text{the run } \pi'|_{\mathcal{A}} \text{ is accepting}$$

- Vice versa, if $\pi = s_0\, \alpha_0\, s_1\, \alpha_1\, s_2\, \alpha_2 \ldots$ is a path in $\mathcal{M}$ and $\rho = q_0\, q_1\, q_2 \ldots$ a run in $\mathcal{A}$ for its trace, then

$$\pi_\rho \;=\; \langle s_0, q_0\rangle\, \gamma_0\, \langle s_1, q_1\rangle\, \gamma_1\, \langle s_2, q_2\rangle\, \gamma_2 \ldots$$

  is a path in $\mathcal{M} \otimes \mathcal{A}$ where $\gamma_i = \langle \alpha_i, q_{i+1}\rangle$. In this case, we have: ρ is accepting iff $\pi_\rho \models \Phi$.

Proof of statement (a). To show (a), we demonstrate that a scheduler for $\mathcal{M} \otimes \mathcal{A}$ that maximizes the probability for Φ can be transferred to a scheduler for $\mathcal{M}$ while maintaining the probabilities when considering the language $\mathcal{L}_\omega(\mathcal{A})$. Intuitively, this holds as the scheduler choices for the $\mathcal{A}$-successors in the product represent a specific resolution of the non-determinism in $\mathcal{A}$. The converse does not necessarily hold in the non-good-for-game case, as the scheduler in $\mathcal{M} \otimes \mathcal{A}$ has to commit to a particular resolution of the non-determinism in $\mathcal{A}$, without being able to predict the future.

We pick a scheduler $\mathfrak{s}'$ for $\mathcal{M} \otimes \mathcal{A}$ that maximizes the probability for $\mathcal{A}$'s acceptance condition. The goal is to derive a scheduler $\mathfrak{s}$ for $\mathcal{M}$ under which the probability for generating traces in $\mathcal{L}_\omega(\mathcal{A})$ is at least

$$\Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}}\big(\,\Phi\,\big) \;=\; \Pr^{\mathfrak{s}'}_{\mathcal{M}\otimes\mathcal{A}}\big(\,\Phi\,\big).$$

The task is now to define $\mathfrak{s}(\pi) \in \mathit{Act}(\,\mathit{last}(\pi)\,)$ for finite paths π in $\mathcal{M}$ where $\mathit{last}(\pi)$ denotes the last state of π. Let $\pi = s_0\, \alpha_0\, s_1\, \alpha_1\, \ldots \alpha_{n-1}\, s_n$ be a finite path in $\mathcal{M}$. We introduce inductively states $q_1, \ldots, q_n, q_{n+1}$ in $\mathcal{A}$ as follows. Let

$$\gamma_0 \;\stackrel{\text{def}}{=}\; \langle \alpha_0, q_1\rangle \;=\; \mathfrak{s}'(\langle s_0, q_0\rangle)$$

and for $0 \leq i \leq n$:

$$\gamma_i \;\stackrel{\text{def}}{=}\; \langle \alpha_i, q_{i+1}\rangle \;=\; \mathfrak{s}'\big(\,\langle s_0, q_0\rangle\, \gamma_0\, \langle s_1, q_1\rangle\, \gamma_1\, \ldots\, \gamma_{i-1}\, \langle s_i, q_i\rangle\,\big)$$

Clearly, in the above inductive definition we have $q_{i+1} \in \delta(q_i, \ell(s_i))$ and $\alpha_i \in \mathit{Act}(s_i)$. We then define:

$$\mathfrak{s}(s_0\, \alpha_0\, s_1\, \alpha_1 \ldots \alpha_{n-1}\, s_n) \;\stackrel{\text{def}}{=}\; \alpha_n$$

Now, we show that for every $\mathfrak{s}'$-path in $\mathcal{M} \otimes \mathcal{A}$ with a trace satisfying the acceptance condition Φ, there exists a $\mathfrak{s}$-path in $\mathcal{M}$ with a trace accepted by $\mathcal{A}$. Suppose that $\pi' = \langle s_0, q_0\rangle \gamma_0 \langle s_1, q_1\rangle \gamma_1 \langle s_2, q_2\rangle \gamma_2 \ldots$ is an infinite $\mathfrak{s}'$-path with $\pi' \models \Phi$. In $\mathcal{M} \otimes \mathcal{A}$, for an action $\gamma_i = \langle \alpha_i, q_{i+1}\rangle$ to be enabled, $q_{i+1} \in \delta(q_i, \alpha_i)$ must hold. Since π' satisfies Φ, $\pi'|_{\mathcal{A}} = q_0\, q_1\, \ldots$ is an accepting run in $\mathcal{A}$ for the trace of $\pi'|_{\mathcal{M}} = s_0\, \alpha_0\, s_1\, \alpha_1\, s_2\, \alpha_2 \ldots$. By definition of $\mathcal{M} \otimes \mathcal{A}$, $\pi'|_{\mathcal{M}} = s_0\, \alpha_0\, s_1 \alpha_1\, s_2\, \alpha_2 \ldots$ is a path in $\mathcal{M}$. Thus, the set of all $\mathfrak{s}$-paths π with $\mathit{trace}(\pi) \in \mathcal{L}_\omega(\mathcal{A})$ contains the set of the paths $\pi'|_{\mathcal{M}}$ where π' is a $\mathfrak{s}'$-path in $\mathcal{M} \otimes \mathcal{A}$ with $\pi' \models \Phi$ and consequently $\Pr^{\mathfrak{s}'}_{\mathcal{M}\otimes\mathcal{A}}(\Phi) \leq \Pr^{\mathfrak{s}}_{\mathcal{M}}(\mathcal{A})$. We obtain the desired result:

$$\Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}}(\Phi) \;=\; \Pr^{\mathfrak{s}'}_{\mathcal{M}\otimes\mathcal{A}}(\Phi) \;\leq\; \Pr^{\mathfrak{s}}_{\mathcal{M}}\big(\mathcal{L}_\omega(\mathcal{A})\big) \;\leq\; \Pr^{\max}_{\mathcal{M}}\big(\mathcal{L}_\omega(\mathcal{A})\big)$$

Proof of statement (b). To show (b), we show that the good-for-games property allows a scheduler in $\mathcal{M} \otimes \mathcal{A}$ to resolve the non-determinism induced by $\mathcal{A}$ in the product in an optimal way. Technically, this relies on combining a maximizing scheduler for $\mathcal{M}$ and $\mathcal{L}_\omega(\mathcal{A})$ with a GFG-strategy for $\mathcal{A}$.

We suppose now that $\mathcal{A}$ is good-for-games. By (a), it suffices to show that

$$\Pr^{\max}_{\mathcal{M}}(\mathcal{A}) \;\leq\; \Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}}(\Phi)$$

Let $\mathfrak{f}$ denote a GFG-strategy for the monitor game for $\mathcal{A}$ and let $\mathfrak{s}$ be a scheduler in $\mathcal{M}$ that maximizes the probability to generate traces in $\mathcal{L}_\omega(\mathcal{A})$. The goal is to compose $\mathfrak{s}$ and $\mathfrak{f}$ to obtain a scheduler $\mathfrak{s}'$ for $\mathcal{M} \otimes \mathcal{A}$ such that the probability under $\mathfrak{s}'$ for the paths π' with $\pi' \models \Phi$ is at least $\Pr^{\mathfrak{s}}_{\mathcal{M}}(\mathcal{A})$.

The definition of $\mathfrak{s}'(\pi')$ for the finite paths π' in $\mathcal{M} \otimes \mathcal{A}$ is by induction on the length of π'. For the initial state we define:

$$\mathfrak{s}'(\langle s_0, q_0\rangle) \;\stackrel{\text{def}}{=}\; \langle\, \mathfrak{s}(s_0),\, \mathfrak{f}(q_0\, \ell(s_0))\,\rangle$$

For a finite path $\pi' = \langle s_0, q_0\rangle \gamma_0 \langle s_1, q_1\rangle \gamma_1 \ldots \gamma_{n-1} \langle s_n, q_n\rangle$ in $\mathcal{M} \otimes \mathcal{A}$ of length $n \geq 1$ where $\gamma_i = \langle \alpha_i, q_{i+1}\rangle$, the definition of $\mathfrak{s}'(\pi')$ is as follows:

$$\mathfrak{s}'(\pi') \;\stackrel{\text{def}}{=}\; \langle\, \mathfrak{s}(\pi'|_{\mathcal{M}}),\, \mathfrak{f}\big(q_0\, \ell(s_0)\, q_1\, \ell(s_1)\, \ldots\, q_n\, \ell(s_n)\big)\,\rangle$$

Suppose now that $\pi = s_0\, \alpha_0\, s_1\, \alpha_1\, s_2\, \alpha_2 \ldots$ is an infinite $\mathfrak{s}$-path in $\mathcal{M}$ with $\mathit{trace}(\pi) \in \mathcal{L}_\omega(\mathcal{A})$. We now consider the accepting run $\rho = q_0\, q_1\, q_2\, \ldots$ for $\mathit{trace}(\pi)$ that is obtained using the GFG-strategy $\mathfrak{f}$ in the monitor game for $\mathcal{A}$. That is:

$$q_{i+1} \;=\; \mathfrak{f}\big(\, q_0\, \ell(s_0)\, q_1\, \ell(s_1)\, \ldots\, q_i\, \ell(s_i)\,\big)$$

Then, $\pi_\rho = \langle s_0, q_0\rangle \gamma_0 \langle s_1, q_1\rangle \gamma_1 \langle s_2, q_2\rangle \gamma_2 \ldots$ is an infinite $\mathfrak{s}'$-path with $\pi_\rho \models \Phi$ where $\gamma_i = \langle \alpha_i, q_{i+1}\rangle$. Thus, the set of all infinite $\mathfrak{s}'$-paths π' with $\pi' \models \Phi$ subsumes all paths π_ρ resulting from combining an $\mathfrak{s}$-path π in $\mathcal{M}$ where $\mathit{trace}(\pi) \in \mathcal{L}_\omega(\mathcal{A})$ with its (unique) accepting $\mathfrak{f}$-run ρ. This yields:

$$\Pr^{\max}_{\mathcal{M}}\big(\mathcal{L}_\omega(\mathcal{A})\big) \;=\; \Pr^{\mathfrak{s}}_{\mathcal{M}}\big(\mathcal{L}_\omega(\mathcal{A})\big) \;\leq\; \Pr^{\mathfrak{s}'}_{\mathcal{M}\otimes\mathcal{A}}(\Phi) \;\leq\; \Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}}(\Phi)$$

This completes the proof of statement (b) in Theorem 3.3. □

Example 3.4 (Necessity of the GFG Property in Theorem 3.3(b))**.** *To illustrate that the GFG-property is crucial in part (b) of Theorem 3.3, we provide an example for an MDP $\mathcal{M}$ (even a Markov chain) and non-deterministic ω-automaton $\mathcal{A}$ such that* $\Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}}(\Phi)$ *is strictly smaller than* $\Pr^{\max}_{\mathcal{M}}(\mathcal{A})$.

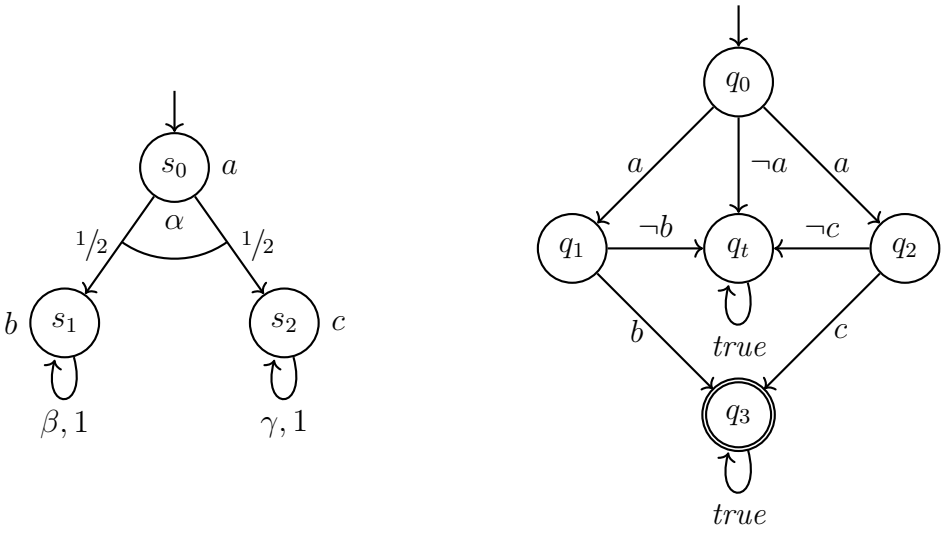


Figure 3.3: Markov decision process $\mathcal{M}$ (left) and NBA $\mathcal{A}$ for $a \wedge \bigcirc(b \vee c)$ (right).

The left of Figure 3.3 shows the MDP $\mathcal{M}$ with three states s_0, s_1, s_2, actions α, β and γ and atomic propositions a, b and c. $\mathcal{M}$ behaves purely probabilistically. Hence, it can be seen as a Markov chain and the concept of schedulers is irrelevant for $\mathcal{M}$. As $\mathcal{M}$ has only two paths $\pi_1 = s_0\,\alpha\,s_1\,\beta\,s_1\,\beta\,s_1\,\beta\ldots$ and $\pi_2 = s_0\,\alpha\,s_2\,\gamma\,s_2\,\gamma\,s_2\,\gamma\ldots$, $\mathcal{M}$ has only two traces, namely $\{a\}\,\{b\}^\omega$ and $\{a\}\,\{c\}^\omega$.

The picture on the right shows the same NBA as in Figure 3.2. It is an NBA $\mathcal{A}$ over the alphabet $2^{\{a,b,c\}}$ and the accepting state q_3. Thus, Φ is given by $\mathrm{Inf}(q_3)$. *Clearly, $\mathcal{L}_\omega(\mathcal{A})$ is the language for the LTL formula*

$$\varphi \;=\; a \wedge \bigcirc(b \vee c)$$

Since both paths π_1 and π_2 of $\mathcal{M}$ satisfy φ, the probability for φ in $\mathcal{M}$ is 1, which yields $\Pr^{\max}_{\mathcal{M}}(\mathcal{A}) = 1$.

Figure 3.4 shows the product-MDP $\mathcal{M}\otimes\mathcal{A}$. Any scheduler $\mathfrak{s}$ for $\mathcal{M}\otimes\mathcal{A}$ just has two options in the initial state $\langle s_0, q_0\rangle$, "select action $\langle\alpha, q_1\rangle$" or "select action $\langle\alpha, q_2\rangle$", while there is just one enabled action for the other states. In particular, $\mathcal{M}\otimes\mathcal{A}$ has just two schedulers.

By symmetry, it suffices to consider the scheduler $\mathfrak{s}$ choosing action $\langle\alpha, q_1\rangle$ for the initial state. The $\mathfrak{s}$-paths resolve the probabilistic choice between $\langle s_1, q_1\rangle$ and $\langle s_2, q_1\rangle$. No accepting state of the form $\langle s, q_3\rangle$ is reachable from state $\langle s_2, q_1\rangle$, while the accepting state $\langle s_1, q_3\rangle$ will be reached in the next step from $\langle s_1, q_1\rangle$. Hence:

$$\Pr^{\mathfrak{s}}_{\mathcal{M}\otimes\mathcal{A}}(\Phi) \;=\; \frac{1}{2}$$

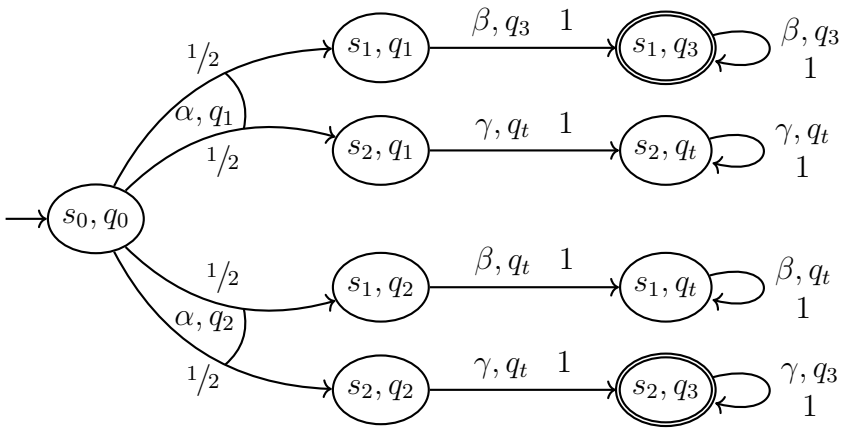


Figure 3.4: Product MDP $\mathcal{M} \otimes \mathcal{A}$.

The same argument applies to the scheduler for $\mathcal{M} \otimes \mathcal{A}$ that chooses action $\langle \alpha, q_2 \rangle$ in the first step. Thus, we get:

$$\frac{1}{2} \;=\; \Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}}(\Phi) \;<\; \Pr^{\max}_{\mathcal{M}}(\mathcal{A}) \;=\; 1$$

Theorem 3.3 (b) shows that with a slightly modified definition of the product, the techniques that are known for the quantitative analysis of MDPs against deterministic ω-automata specifications [Var85; VW86; Alf97] are also applicable for GFG automata. The computation of maximal probabilities for properties given by a standard ω-regular acceptance condition Φ (e.g., Büchi, Rabin, parity or Streett) can be carried out by a graph analysis that replaces Φ with a reachability condition and linear programming techniques for computing maximal reachability probabilities. See, e.g., [BK08; BGC09a; BGC09b]. The time complexity is polynomial in the size of $\mathcal{M}$ and $\mathcal{A}$, matching the complexity of the approach using deterministic ω-automata.

[KV05; KR11] proves that a double-exponential blow-up for translating LTL to deterministic ω-automata (of any type) is unavoidable. We show now how the proof in [KR11] can be adapted for GFG automata, yielding a double-exponential lower bound for GFG automata as well, which is in accordance with the known 2EXPTIME-completeness for the analysis of MDPs against LTL specifications [CY95].

Theorem 3.5. *There exists a family of LTL formulas $(\varphi_n)_{n\in\mathbb{N}}$ such that $|\varphi_n| = \mathcal{O}(n)$, while every GFG Emerson-Lei automaton $\mathcal{A}_n$ for φ_n has at least $2^{2^{\Omega(\sqrt{n})}}$ states.*

Proof. The proof relies on the following family of ω-regular languages $(L_n)_{n\in\mathbb{N}}$ for which a double-exponential blow-up is unavoidable. Let $\Sigma = \{a, b\}$ be an alphabet with two elements. [KV05] defines a family of languages L_n over the alphabet $\Sigma \cup \{\#, \$\}$ of the following form:

$$L_n \;=\; \bigcup_{w\in\Sigma^n} L(w)$$

where $L(w)$ is the following ω-regular language:

$$L(w) \;=\; \Big\{\, w_1 \# \,\ldots\, \# \, w_{i-1} \# w \# \, w_{i+1} \# \,\ldots\, \# w_m \# \$ \, w \, \#^\omega \;:\; w_1, \ldots, w_{i-1}, w_{i+1}, \ldots, w_m \in \Sigma^*,\; m \in \mathbb{N}, 0 \leqslant i \leqslant m \,\Big\}$$

The languages L_n are safety languages and can be recognized by a DBA $\mathcal{A}_n$, which are known to need at least $2^{2^{\Omega(n)}}$ states. Intuitively, this is because $\mathcal{A}_n$ needs to remember all words of length n that have been seen before \$ to check whether one of them matches the word after \$. There exist 2^n possible words of length n and therefore 2^{2^n} possibilities for which words of length n have occurred before the \$-symbol. Thus, intuitively, for checking whether a duplicated word of length n occurs after \$, $\mathcal{A}_n$ needs at least $2^{2^{\Omega(n)}}$ states. The proof from [KV05], that every DBA accepting the language L_n needs at least $2^{2^{\Omega(n)}}$ states can be extended to a GFG automata in a straightforward manner.

We will prove the claim that every GFG automaton recognizing L_n has at least $2^{2^{\Omega(n)}}$ states by contradiction. We refer to words in Σ^n as n-blocks. Further we assume a total order on Σ^n, thereby we can define the n-block w_i as the i-th word in Σ^n. In this proof we will use index sets containing indices ranging from 1 to 2^n and corresponding to the n-blocks that have appeared before the \$. We define for every such index set

$$I \;=\; \big\{i_1, i_2, \ldots, i_k\big\} \;\subseteq\; \big\{0, 1, \ldots, 2^n{-}1\big\}$$

the finite word $w_I = \#w_{i_1}\#w_{i_2}\#\ldots\#w_{i_k}\#\$$. As $\mathcal{A}_n$ is good-for-games, there exists a GFG-strategy $\mathfrak{f}$. Let q_I be the state reached in the $\mathfrak{f}$-play after consuming the finite word w_I. Since there are 2^n different n-blocks, there exist 2^{2^n} subsets of $\{0, 1, \ldots, 2^n{-}1\}$, but by assumption $\mathcal{A}_n$ has less than 2^{2^n} states. Therefore there must be two distinct sets $I \neq J$ with $q_I = q_J$. W.l.o.g. let $i \in I \setminus J$. The word $w_I \, w_i \, \#^\omega$ belongs to L_n, and hence is accepted by $\mathcal{A}_n$ and the $\mathfrak{f}$-play for $w_I \, w_i \, \#^\omega$ is accepting as well. On the other hand, $w_J \, w_i \, \#^\omega$ is not in L_n. But as $q_I = q_J$ there is an accepting run for the suffix $w_i \, \#^\omega$ and we get $w_J \, w_i \, \#^\omega \in L_n$. Contradiction.

As other standard acceptance conditions like Rabin, parity or Streett are a particular form of Emerson-Lei, the same argument holds also for GFG Rabin, GFG parity, GFG Streett automata and GFG Emerson-Lei automata. As shown in [KV05], there is an LTL formula φ_n of size $\mathcal{O}\big(n^2\big)$ for L_n yielding the double-exponential lower bound for the transformation from LTL to GFG automata. □

Minimal probabilities If $\mathcal{M}$ is an MDP as before and $\mathfrak{s}$ a scheduler for $\mathcal{M}$ then for each ω-regular language L over the alphabet 2^{AP}:

$$\Pr^{\mathfrak{s}}_{\mathcal{M}}(L) \;=\; 1 - \Pr^{\mathfrak{s}}_{\mathcal{M}}(\overline{L})$$

where $\overline{L}$ denotes the complement of L, i.e., $\overline{L} = \big(2^{AP}\big)^\omega \setminus L$. Hence, we get:

$$\Pr^{\min}_{\mathcal{M}}(L) \;=\; 1 - \Pr^{\max}_{\mathcal{M}}(\overline{L})$$

As a consequence of Theorem 3.3, we obtain:

Corollary 3.6. *For each MDP $\mathcal{M}$, LTL formula φ and complete GFG-automaton $\mathcal{A}$ with acceptance condition Φ for $\neg\varphi$:*

$$\Pr^{\min}_{\mathcal{M}}(\varphi) \quad = \quad 1 - \Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}}(\Phi)$$

Moreover, $\Pr^{\min}_{\mathcal{M}}(\varphi) \leq 1 - \Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}}(\Phi)$ for each non-deterministic ω-automaton $\mathcal{A}$ for $\neg\varphi$.

3.2 From LTL to GFG Automata

It has previously been shown [KB06; KB07] that it is possible in practice, using the tool `ltl2dstar`, to obtain deterministic ω-automata for a wide range of LTL formula φ via the translation to an NBA and Safra's determinization construction [Saf88] refined by various heuristics. Here, we are interested in replacing Safra's determinization algorithm with the HP-algorithm [HP06] to generate a GFG automaton instead of a deterministic automaton. We first provide an outline of the HP-algorithm and then explain a few new heuristics.

3.2.1 The HP-algorithm

The HP-algorithm transforms an NBA $\mathcal{B} = (Q, \Sigma, \delta, q_0, \Phi)$ with $\Phi = \mathrm{Inf}(F)$ and $|Q| = n$ states into a GFG automaton $\mathcal{A}$ with parity acceptance and at most $2^n \cdot n^{2n}$ states and $2n$ parity colors (or an NRA with n Rabin pairs), which improves on the upper bound given for Safra's determinization algorithm.

Like Safra's construction, the HP-algorithm relies on the simultaneous tracking of multiple subset constructions to determine acceptance or rejection in the NBA. However, while the states of Safra's DRA organize the subsets in trees, the HP-algorithm uses a simpler, linear arrangement of the subsets. The state space $P = (2^Q \times 2^Q)^n$ of the GFG automaton $\mathcal{A}$ consists of n pairs of subsets of NBA states Q, i.e., states of the form $p = \langle (A_1, B_1), \ldots, (A_n, B_n) \rangle$ where $B_i \subseteq A_i \subseteq Q$, plus the following two additional constraints on the state space:

- if $A_i = \varnothing$ for $i < n$, then $A_{i+1} = \varnothing$, and
- for all $i < j$, $A_i \cap A_j = \varnothing$ or $A_j \subseteq B_i$.

The first constraint ensures that an empty A_i implies that all the following A_j are empty as well. The second constraint ensures that each set A is contained in the B-part of some previous pair and disjoint from all sets between the two. Let $T \subseteq P$ be the tuples that satisfy the constraints above. Each set B_i serves to mark those states in A_i that were reached via some accepting state in F of the NBA. The initial state $q_{0,\mathcal{G}}$ of $\mathcal{A}$ is the tuple $\langle (\{q_0\}, \{q_0\} \cap F), (\varnothing, \varnothing), \ldots, (\varnothing, \varnothing) \rangle$, i.e., where A_1 corresponds to the initial state q_0 of the NBA and B_1 is non-empty if and only if q_0 is accepting.

The successor state in $\mathcal{A}$ for symbol σ is obtained by applying the transition function δ of the NBA to each of the subsets and adding states in F to the B_i subsets. In crucial difference to Safra's construction, the HP-algorithm however then introduces significant non-determinism by allowing $\mathcal{A}$ to discard an arbitrary number of states in any of the subsets. For $p = \langle \ldots (A_i, B_i) \ldots \rangle$, the set A'_i in a σ-successor p' of $\mathcal{A}$ thus does not correspond to $A'_i = \delta(A_i, \sigma)$ but there is a non-deterministic choice between any A'_i satisfying $A'_i \subseteq \delta(A_i, \sigma)$, including the empty set.

For the formal definition of the transition function of the GFG automaton $\mathcal{A}$, we rely on the functions $succ_i(p, \sigma) \colon P \times \Sigma \to 2^{(2^Q \times 2^Q)}$ that provide the possible values for the i-th pair (A'_i, B'_i) in the successor state, given a state $p = \langle (A_1, B_1), \ldots, (A_n, B_n) \rangle \in P$ in the GFG automaton $\mathcal{A}$ and symbol σ. The definition of $succ_i$ distinguishes three cases. In the first case, where $A_i \neq B_i \neq \varnothing$ holds, we have

$$succ_i(p, \sigma) = \{(A', B') \ : A' \subseteq \delta(A_i, \sigma) \text{ and } B' \subseteq (\delta(B_i, \sigma) \cap A') \cup (A' \cap F)\}$$

This reflects a "standard" step, where the subset construction proceeds as normal and where B_i is updated with newly visited accepting states. In the second case, whenever $A_i = B_i \neq \varnothing$, we have

$$succ_i(p, \sigma) = \{(A', B') \ : A' \subseteq \delta(A_i, \sigma) \text{ and } B' \subseteq A' \cap F\}$$

As A_i and B_i are equal, this reflects the situation where all states tracked by A_i could have been reached while visiting some accepting state. The remaining third case arises whenever A_i is empty. Here, $succ_i$ allows for A_i to be "reset", starting the subset construction anew by choosing some subset of states of the first set A_1:

$$succ_i(p, \sigma) = \{(A', B') \ : A' \subseteq \delta(A_1, \sigma) \text{ and } B' \subseteq A'\}$$

The transition function $\delta_{\mathcal{G}}$ of $\mathcal{A}$ is then defined by the application of $succ_i$ to each pair, additionally constraining the successor state to those tuples that satisfy the constraints on the state space by restricting to the states in T:

$$\delta_{\mathcal{G}}(p, \sigma) = T \cap \prod_{i=1}^{n} succ_i(p, \sigma)$$

In the acceptance condition of $\mathcal{A}$, the "resets" (whenever $A_i = \varnothing$ for some pair i) are reflected as "bad" events for the pair i, as they signify that the previously tracked runs terminated. The "good" events for pair i in the acceptance condition occur whenever all states in an A_i are marked as having recently visited F, i.e., whenever $A_i = B_i \neq \varnothing$. Infinitely many "good" events for a pair i without "bad" events for that pair then correspond to the existence of an accepting run in the NBA $\mathcal{B}$. This can be straightforwardly encoded as a Rabin condition with n Rabin pairs or as a parity condition by giving the good and bad events of pair i higher priority than the events of pair j for $j > i$.

Formally, a parity acceptance condition for $\mathcal{A}$ is obtained as follows. For a given state $p = \langle (A_1, B_1), \ldots, (A_n, B_n) \rangle$, we obtain the left-most "bad" pair via the function $ind_U(p)$ and the left-most "good" pair via the function $ind_L(p)$, defined as follows:

$$ind_U(p) = \min\left(\{k \,:\, k \in \{2, \ldots, n\} \wedge A_k = \varnothing\} \cup \{n+1\}\right)$$
$$ind_L(p) = \min\left(\{k \,:\, k \in \{2, \ldots, n\} \wedge A_k = B_k \neq \varnothing\} \cup \{n+1\}\right)$$

The value $n+1$ signals that none of the pairs were "bad" or "good", respectively. The coloring function for the parity acceptance condition (with colors $\{0, \ldots, 2n-1\}$) is then obtained via

$$col(p) = \begin{cases} 0 & \text{if } A_1 = B_1 \neq \varnothing \\ 2i+1 & \text{if } ind_U(p) = i+2 \ \wedge \ ind_L(p) \geq i+2 \\ 2i+2 & \text{if } ind_L(p) = i+2 \ \wedge \ ind_U(p) > i+2 \end{cases}$$

An equivalent Rabin acceptance can be easily obtained by rewriting the parity acceptance, as usual.

The HP-algorithm relies on the GFG-strategy to resolve the non-determinism in the constructed automaton $\mathcal{A}$, i.e., which states in the subsets are kept, which are dropped and when to reset. There is a large amount of non-determinism and a lot of combinatorial possibilities in the reachable state space of $\mathcal{A}$. This is confirmed by our experiments, e.g., applying the construction to the two-state NBA for $\Diamond\Box a$ already yields a GFG automaton with 16 states, where `ltl2dstar` generates a two-state DRA. As stated in [HP06], the HP-algorithm is thus not well-suited for an explicit representation for $\mathcal{A}$, but is intended for a symbolic implementation. In this context, [HP06] briefly discusses the possibility of variants of the transition function in the GFG automaton that either apply more or less strict constraints on the relationship enforced between the (A_i, B_i) pairs in each state. Especially, [HP06] posits that introducing even further non-determinism (and increasing the number of possible states) by loosening a disjunctness requirement on the A_i may lead to a smaller symbolic representation. In our experiments, we will consider such a variant of the HP-algorithm, where the second constraint on the state space ("for all $i < j$, $A_i \cap A_j = \varnothing$ or $A_j \subseteq B_i$") is removed. We will refer to this variant as the *loose variant*.

3.2.2 Iterative approach

In the context of games, [HP06] proposes an iterative approach to the HP-algorithm by successively constructing the automata $\mathcal{A}^m$ obtained by using only the first m of the n pairs, i.e., by setting $A_i = B_i = \varnothing$ for all $m < i \leq n$. In the acceptance condition this reduces the number of required parity colors to $2m$ and Rabin pairs to m as well. For these automata, $\mathcal{L}_\omega(\mathcal{A}^m) = \mathcal{L}_\omega(\mathcal{A}) = \mathcal{L}_\omega(\mathcal{B})$, but there is no guarantee that $\mathcal{A}^m$ for $m < n$ is good-for-games by construction. We start with $m = 1$ and increase m until early success or reaching $m = n$. Our experimental results indeed show that early termination appears rather often.

We now explain how the iterative approach of [HP06] can be integrated in the GFG-based quantitative analysis of MDPs against LTL specifications. Suppose, e.g., that the task is to show that $\Pr^{\max}_{\mathcal{M}}(\varphi) \geq \theta$ for some LTL formula φ and threshold $\theta \in]0,1]$. Let $\mathcal{B}$ be an n-state NBA with $\mathcal{L}_\omega(\mathcal{B}) = \mathcal{L}(\varphi)$ and $\mathcal{A}^m$ the automaton obtained using only the first $m \leq n$ pairs in the HP-algorithm applied to $\mathcal{B}$. Let Φ^m denote the acceptance condition of $\mathcal{A}^m$. By Theorem 3.3 (a):

$$\text{If } \Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}^m}(\Phi^m) \geq \theta \text{ for some } m \leq n \text{ then } \Pr^{\max}_{\mathcal{M}}(\varphi) \geq \theta.$$

Moreover, $\Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}^m}(\Phi^m) \leq \Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}^{m+1}}(\Phi^{m+1})$ for $m < n$, since the paths in $\mathcal{M}\otimes\mathcal{A}^m$ constitute a subset of the paths in $\mathcal{M}\otimes\mathcal{A}^{m+1}$. These observations suggest an approach that resembles the classical *abstraction-refinement* schema: starting with $m = 1$, we carry out the quantitative analysis of $\mathcal{M}\otimes\mathcal{A}^m$ against Φ^m and successively increase m until $\Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}^m}(\Phi^m) \geq \theta$ or $\mathcal{A}^m$ is GFG (which is the case at the latest when $m = n$). As an additional heuristic to increase the performance of the linear programming techniques that are applied for the quantitative analysis of $\mathcal{M} \otimes \mathcal{A}^m$ against Φ^m, one can reuse the results computed for $\mathcal{M} \otimes \mathcal{A}^{m-1}$ and Φ^{m-1} as initial values.

GFG checking. It remains to explain how to check whether $\mathcal{A}^m$ has the GFG property for $m < n$, i.e., when it is not clear from the construction itself that the GFG property holds. In this aspect, our prototype implementation departs from [HP06] and checks whether $\mathcal{A}^m$ is GFG by solving a Rabin game (itself an NP-complete problem) constructed from $\mathcal{A}^m$ and a DRA for $\neg\varphi$ constructed with `ltl2dstar` while [HP06] proposes an algorithm based on checking fair simulation. We will use this GFG check in our experimental evaluation to study the impact of the iterative approach in terms of the number of required iterations that are actually needed in practice as well as the size of the resulting GFG automata. To obtain these results, the particular choice of the GFG test is irrelevant.

We detail here a game-based characterization of the GFG property, which serves as the basis for checking whether a given ω-automaton $\mathcal{A}$ is good-for-games in our implementation of the iterative approach of the HP-construction. This approach is based on the game-based approach to determinization of GFG automata presented in [Bok+13].

Given an NRA $\mathcal{A} = (Q, \Sigma, \delta_\mathcal{A}, q_0, \Phi_\mathcal{A})$ then – by definition – $\mathcal{A}$ is good-for-games if there is a strategy that generates an accepting run for exactly the words w with $w \in \mathcal{L}_\omega(\mathcal{A})$. Since the class of ω-regular languages is closed under complementation there exists a DRA $\mathcal{D} = (P, \Sigma, \delta_\mathcal{D}, p_0, \Phi_\mathcal{D})$ with $\mathcal{L}_\omega(\mathcal{D}) = \Sigma^\omega \setminus \mathcal{L}_\omega(\mathcal{A})$. W.l.o.g. we assume transition relations of $\mathcal{A}$ and $\mathcal{D}$ to be total.

We now construct a turn-based two-player game $\mathcal{G}_{\mathcal{A},\mathcal{D}}$ with full observation for both players as follows. The set of game vertices is $V = V_1 \cup V_0$ where the set of vertices where player 1 moves is $V_1 = Q \times P$ and where player 0 moves in the vertices in $V_0 = Q \times P \times \Sigma$. We set the initial vertex to $\langle q_0, p_0\rangle$. The moves in $\mathcal{G}_{\mathcal{A},\mathcal{D}}$ are defined by the following two structural operational semantics rules (SOS-rules):

$$\frac{q \in Q, p \in P, \sigma \in \Sigma}{\langle q, p\rangle \longrightarrow \langle q, p, \sigma\rangle}$$

and

$$\frac{q' \in \delta_{\mathcal{A}}(q,\sigma),\ p' = \delta_{\mathcal{D}}(p,\sigma)}{\langle q,p,\sigma\rangle \longrightarrow (q',p')}$$

Player 1 chooses a symbol $\sigma \in \Sigma$ and player 0 resolves the non-determinism. As $\mathcal{D}$ is deterministic, the game-structure $\mathcal{G}_{\mathcal{A},\mathcal{D}}$ can be viewed as a refinement of the monitor game associated with $\mathcal{A}$. The states in $\mathcal{A}$ are simply augmented with the information on $\mathcal{D}$'s current state and the chosen symbol.

The objective in $\mathcal{G}_{\mathcal{A},\mathcal{D}}$ is defined such that player 0 wins a play ς, if for every word w:

- if $w \in \mathcal{L}_\omega(\mathcal{A})$, then $\varsigma|_{\mathcal{A}}$ is an accepting run in $\mathcal{A}$
- if $w \notin \mathcal{L}_\omega(\mathcal{A})$, then $\varsigma|_{\mathcal{D}}$ is an accepting run in $\mathcal{D}$.

where $\varsigma|_{\mathcal{A}}$ denotes the projection of ς to the states $\mathcal{A}$. More precisely, we erase all vertices $\langle q,p,\sigma\rangle$ from ς and replace the vertices $\langle q,p\rangle$ with q. Likewise $\varsigma|_{\mathcal{D}}$ arises from ς by taking the projection to the $\mathcal{D}$-components of the vertices $\langle q,p\rangle$ in ς. Formally, the objective for player 0 in the game $\mathcal{G}_{\mathcal{A},\mathcal{D}}$ is the Rabin condition resulting from the union of the acceptances of $\mathcal{A}$ and $\mathcal{D}$ lifted to the product. That is, if

$$\begin{aligned}\Phi_{\mathcal{A}} &= \bigvee_{1\le i\le n} \mathrm{Fin}\left(U_i^{\mathcal{A}}\right) \wedge \mathrm{Inf}\left(L_i^{\mathcal{A}}\right)\\ \Phi_{\mathcal{D}} &= \bigvee_{1\le i\le m} \mathrm{Fin}\left(U_i^{\mathcal{D}}\right) \wedge \mathrm{Inf}\left(L_i^{\mathcal{D}}\right)\end{aligned}$$

are the Rabin acceptance of $\mathcal{A}$ and $\mathcal{D}$, respectively, then we define the objective for player 0 in the game $\mathcal{G}_{\mathcal{A},\mathcal{D}}$ as the Rabin acceptance

$$\Psi = \bigvee_{1\le i\le n+m} \mathrm{Fin}\left(U_i\right) \wedge \mathrm{Inf}\left(L_i\right)$$

where for $1 \le i \le n$ and $1 \le j \le m$:

$$\begin{aligned}U_i &= \{\langle q,p\rangle \in V_0 : q \in U_i^{\mathcal{A}}\} & U_{n+j} &= \{\langle q,p\rangle \in V_0 : q \in U_j^{\mathcal{B}}\}\\ L_i &= \{\langle q,p\rangle \in V_0 : q \in L_i^{\mathcal{A}}\} & L_{n+j} &= \{\langle q,p\rangle \in V_0 : q \in L_j^{\mathcal{B}}\}\end{aligned}$$

Lemma 3.7 (Game-based characterization of the GFG property)**.** *$\mathcal{A}$ is good-for-games iff player 0 has a winning strategy in the Rabin game $\mathcal{G}_{\mathcal{A},\mathcal{D}}$.*

In what follows, let $\mathcal{G} = \mathcal{G}_{\mathcal{A},\mathcal{D}}$.

Proof of "$\Longleftarrow$" Assume that player 0 has a winning strategy $\mathfrak{g} : (V_1\, V_0)^+ \to V_1$ in $\mathcal{G}$. To define a GFG-strategy $\mathfrak{f} : (Q \times \Sigma)^+ \to Q$ for player 0 in the monitor game for $\mathcal{A}$, we first look at the play fragment $q_0\,\sigma_1\,q_1\,\sigma_2\,\ldots\,q_n\,\sigma_{n+1}$ and consider the choice of $\mathfrak{g}$ in $\mathcal{G}$ for the following play fragment in $\mathcal{G}$:

$$\varsigma = \langle q_0,p_0\rangle\,\langle q_0,p_0,\sigma_1\rangle\,\langle q_1,p_1\rangle\,\langle q_1,p_1,\sigma_2\rangle\,\ldots\,\langle q_n,p_n\rangle\,\langle q_n,p_n,\sigma_{n+1}\rangle$$

where $p_i = \delta_{\mathcal{D}}(p_0, \sigma_0\,\sigma_1 \ldots \sigma_{i-1})$ is the unique state in $\mathcal{D}$ that is reached from p_0 by reading the finite input string $\sigma_0\,\sigma_1 \ldots \sigma_{i-1}$. Let $\langle q_{n+1}, p_{n+1}\rangle = \mathfrak{g}(\varsigma)$. Then, we define:

$$\mathfrak{f}(q_0\,\sigma_1\,q_2\,\sigma_2\,\ldots\,q_n\,\sigma_n) \;=\; q_{n+1}$$

Suppose that $w = \sigma_0\,\sigma_1\,\sigma_2\,\ldots \in \mathcal{L}_\omega(\mathcal{A})$. Since the strategy $\mathfrak{g}$ is winning, the $\mathfrak{g}$-play induced by w,

$$\varsigma \;=\; \langle q_0, p_0\rangle\,\langle q_0, p_0, \sigma_1\rangle\,\langle q_1, p_1\rangle\,\langle q_1, p_1, \sigma_2\rangle\,\ldots,$$

in $\mathcal{G}$ is winning for player 0, i.e., ς satisfies the Rabin condition Ψ associated with $\mathcal{G}$. We pick some Rabin pair $\mathrm{Fin}\,(U_i) \wedge \mathrm{Inf}\,(L_i)$ with $1 \leq i \leq n$ such that $\varsigma \models \mathrm{Fin}\,(U_i)$ and $\varsigma \models \mathrm{Inf}\,(L_i)$. There has to exist such a Rabin pair, since ς has to satisfy the winning objective which means that for an accepted word $w \in \mathcal{L}_\omega(\mathcal{A})$ we have to fulfill a Rabin pair stemming from $\mathcal{A}$. By taking the projection of all states in $\mathrm{Fin}\,(U_i) \wedge \mathrm{Inf}\,(L_i)$ we obtain a Rabin pair $\mathrm{Fin}\,\big(U_i^{\mathcal{A}}\big) \wedge \mathrm{Inf}\,\big(L_i^{\mathcal{A}}\big)$ in $\mathcal{A}$ and the $\mathfrak{f}$-play induced by w is $q_0\,\sigma_1\,q_1\,\sigma_2\ldots$. Hence, $\varsigma|_{\mathcal{A}} \;=\; q_0\,q_1\,\ldots$ is a run for w in $\mathcal{A}$ and

$$\varsigma|_{\mathcal{A}} \models \mathrm{Fin}\,\big(U_i^{\mathcal{A}}\big) \quad \text{and} \quad \varsigma|_{\mathcal{A}} \models \mathrm{Inf}\,\big(L_i^{\mathcal{A}}\big).$$

Thus, ς meets the Rabin condition of $\mathcal{A}$.

Proof of "$\Longrightarrow$" Assume $\mathcal{A}$ is good-for-games. Then, there exists a GFG-strategy $\mathfrak{f} : (Q \times \Sigma)^+ \to Q$ for the monitor game for $\mathcal{A}$. We define the strategy $\mathfrak{g}$ for player 0 in $\mathcal{G}$ as follows. Given the play fragment

$$\varsigma \;=\; \langle q_0, p_0\rangle\,\langle q_0, p_0, \sigma_1\rangle\,\langle q_1, p_1\rangle\,\langle q_1, p_1, \sigma_2\rangle\,\ldots\,\langle q_n, p_n\rangle\,\langle q_n, p_n, \sigma_n\rangle$$

in $\mathcal{G}$ we define $\mathfrak{g}(\varsigma)$ as follows:

$$\mathfrak{g}(\varsigma) \quad = \quad \langle \mathfrak{f}(q_0\,\sigma_1\,q_1\,\sigma_2\,\ldots\,q_n\,\sigma_n), \delta_{\mathcal{D}}(p_n, \sigma_n)\rangle$$

Let $w = \sigma_0\,\sigma_1\,\sigma_2\ldots \in \Sigma^\omega$ be an infinite word.
Case 1: $w \in \mathcal{L}_\omega(\mathcal{A})$. For the $\mathfrak{g}$-play $\varsigma \;=\; \langle q_0, p_0\rangle\,\langle q_0, p_0, \sigma_0\rangle\,\langle q_1, p_1\rangle\,\langle q_1, p_1, \sigma_1\rangle\,\ldots$ induced by w in $\mathcal{G}$ we have:

$$\varsigma|_{\mathcal{A}} \;=\; q_0\,q_1\,q_2\,\ldots \;\models\; \mathrm{Fin}\,\big(U_i^{\mathcal{A}}\big) \;\wedge\; \mathrm{Inf}\,\big(L_i^{\mathcal{A}}\big)$$

for some $i \in \{1, \ldots, n\}$. As $\mathcal{G}$'s objective Ψ contains the corresponding Rabin pair (U_i, L_i), we get $\varsigma \models \Psi$.
Case 2: $w \notin \mathcal{L}_\omega(\mathcal{A})$. Then, $w \in \mathcal{L}_\omega(\mathcal{D})$. Let ρ be the unique run for w in $\mathcal{D}$. Then, $\rho \models \Phi_{\mathcal{D}}$. Hence, there exists $j \in \{1, \ldots, m\}$ with $\rho \models \mathrm{Fin}\,\big(U_j^{\mathcal{D}}\big)$ and $\rho \models \mathrm{Inf}\,\big(L_j^{\mathcal{D}}\big)$. Let

$$\varsigma \;=\; \langle q_0, p_0\rangle\,\langle q_0, p_0, \sigma_0\rangle\,\langle q_1, p_1\rangle\,\langle q_1, p_1, \sigma_1\rangle\,\langle q_2, p_2\rangle\,\langle q_2, p_2, \sigma_2\rangle\,\ldots$$

be the $\mathfrak{g}$-play in $\mathcal{G}$ if player 1 chooses the symbols σ_i according to w. Then, $\varsigma|_{\mathcal{D}} = \rho$. Hence, $\varsigma \models \mathrm{Fin}\,(U_{n+j})$ and $\varsigma \models \mathrm{Inf}\,(L_{n+j})$. Thus, ς satisfies the objective Ψ for player 0 in $\mathcal{G}$. □

In practice, when starting from an LTL formula φ, we do not construct $\mathcal{D}$ from $\mathcal{A}$ via complementation and determinization. As we know that $\mathcal{L}_\omega(\mathcal{A}) = \mathcal{L}(\varphi)$, we obtain the DRA $\mathcal{D}$ by applying standard algorithms for the construction of a DRA for the negation $\neg\varphi$, i.e., via the tool `ltl2dstar`. Even though $\mathcal{A}$ and $\mathcal{D}$ (and thus the game as well) have a worst-case double-exponential number of states in the size of the formula, and though solving Rabin games is itself NP-complete, we have been able to use this approach in practice for the smaller automata (see Section 3.3) to determine whether the intermediate automata in the iterative approach of the HP-construction are GFG or not.

3.2.3 Union operator for disjunctive formulas

For generating a deterministic automaton from an LTL formula, it has been shown in [KB06] that optionally handling disjunctive LTL formulas of the form $\varphi = \varphi_1 \vee \varphi_2$ by constructing DRA $\mathcal{A}_1$ and $\mathcal{A}_2$ for the subformulas φ_1 and φ_2 and then obtaining the DRA $\mathcal{A}_1 \cup \mathcal{A}_2$ for the language $\mathcal{L}_\omega(\mathcal{A}_1) \cup \mathcal{L}_\omega(\mathcal{A}_2)$ via a product construction can be very beneficial in practice. The definition of $\mathcal{A}_1 \cup \mathcal{A}_2$ used in [KB06] can easily be extended to NRA. The GFG property is preserved by the union construction.

Definition 3.8 (Union of two NRA)**.** *Let* $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_{0,1}, \Phi_1)$ *and* $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, q_{0,2}, \Phi_2)$ *be two complete NRA over the same alphabet with* $\Phi_1 = \big(\mathrm{Fin}\,(U_{1,1}) \wedge \mathrm{Inf}\,(L_{1,1})\big) \vee \ldots \vee \big(\mathrm{Fin}\,(U_{1,n}) \wedge \mathrm{Inf}\,(L_{1,n})\big)$ *and* $\Phi_2 = \big(\mathrm{Fin}\,(U_{2,1}) \wedge \mathrm{Inf}\,(L_{2,1})\big) \vee \ldots \vee \big(\mathrm{Fin}\,(U_{2,n}) \wedge \mathrm{Inf}\,(L_{2,n})\big)$*. The NRA* $\mathcal{A}_1 \cup \mathcal{A}_2 = (Q', \Sigma, \delta', q_0', \Phi')$ *is defined as follows. The state space of* $\mathcal{A}_1 \cup \mathcal{A}_2$ *is* $Q' = Q_1 \times Q_2$ *and* $q_0' = (q_{0,1}, q_{0,2})$ *its initial state. The transition function* δ' *is given by:*

$$\delta'((q_1, q_2), \sigma) \;=\; \big\{\, (q_1', q_2') : q_1' \in \delta_1(q_1, \sigma),\, q_2' \in \delta_2(q_2, \sigma) \,\big\}$$

The Rabin acceptance Φ' *is given by:*

$$\bigvee_{1 \le i \le n} \big(\, \mathrm{Fin}\,(U_{1,i} \times Q_2) \,\wedge\, \mathrm{Inf}\,(L_{1,i} \times Q_2) \,\big) \quad \vee \bigvee_{1 \le j \le m} \big(\, \mathrm{Fin}\,(Q_1 \times U_{2,j}) \,\wedge\, \mathrm{Inf}\,(Q_1 \times L_{2,j}) \,\big)$$

Obviously, $\mathcal{L}_\omega(\mathcal{A}_1 \cup \mathcal{A}_2) = \mathcal{L}_\omega(\mathcal{A}_1) \cup \mathcal{L}_\omega(\mathcal{A}_2)$. Additionally, the union operation preserves the GFG property.

Lemma 3.9 (Good-for-games for the union operation)**.** *Let* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *be complete NRA. If* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *are GFG, then* $\mathcal{A}_1 \cup \mathcal{A}_2$ *is good-for-games, too.*

Proof. Let $\mathfrak{f}_1 : (Q_1 \times \Sigma)^+ \to Q_1$ be a GFG-strategy for $\mathcal{A}_1$ and $\mathfrak{f}_2 : (Q_2 \times \Sigma)^+ \to Q_2$ be a GFG-strategy for $\mathcal{A}_2$. We define a strategy

$$\mathfrak{f} : (Q' \times \Sigma)^+ \to Q'$$

for $\mathcal{A}_1 \cup \mathcal{A}_2$ and $\varsigma = \langle q_{0,1}, q_{0,2} \rangle\, \sigma_0 \ldots \langle q_{i,1}, q_{i,2} \rangle\, \sigma_i$ as follows:

$$\mathfrak{f}(\varsigma) \;\stackrel{\text{def}}{=}\; \langle\, \mathfrak{f}_1(\varsigma\big|_1),\, \mathfrak{f}_2(\varsigma\big|_2) \,\rangle$$

where $\varsigma\big|_1 = q_{0,1}\,\sigma_0\,q_{1,1}\,\ldots\,q_{i-1,1}\sigma_i$ denotes the projection of the play ς to the first automaton. $\varsigma\big|_2$ is defined analogously.

The goal is to show that $\mathfrak{f}$ is a GFG-strategy for $\mathcal{A}_1 \cup \mathcal{A}_2$. Let $w = \sigma_0\,\sigma_1\,\sigma_2\,\ldots \in \mathcal{L}_\omega(\mathcal{A}_1 \cup \mathcal{A}_2) = \mathcal{L}_\omega(\mathcal{A}_1) \cup \mathcal{L}_\omega(\mathcal{A}_2)$ and let

$$\varsigma \;=\; \langle q_{0,1}, q_{0,2}\rangle\,\sigma_0\,\langle q_{1,1}, q_{1,2}\rangle\,\sigma_1\,\langle q_{2,1}, q_{2,2}\rangle\,\sigma_2\ldots$$

be the induced $\mathfrak{f}$-play in the monitor game for $\mathcal{A}_1 \cup \mathcal{A}_2$ with $\varsigma\big|_\Sigma = w$. W.l.o.g. we may suppose that $w \in \mathcal{L}_\omega(\mathcal{A}_1)$. By the definition of $\mathfrak{f}$, the play $\varsigma\big|_1 = q_{0,1}\,\sigma_0\,q_{1,1}\,\sigma_1\,q_{2,1}\,\sigma_2\ldots$ in the monitor game for $\mathcal{A}_1$ is $\mathfrak{f}_1$-conform, and hence $q_{0,1}\,q_{1,1}\,q_{2,1}\ldots$ is an accepting run in $\mathcal{A}_1$. Thus, there is a Rabin pair $\mathrm{Fin}\,(U_w) \wedge \mathrm{Inf}\,(L_w)$ in Φ_1 with

$$q_{0,1}\,q_{1,1}\,\ldots \;\models\; \mathrm{Fin}\,(U_w) \wedge \mathrm{Inf}\,(L_w)$$

Hence, for the run $\varsigma\big|_Q = \langle q_{0,1}, q_{0,2}\rangle\,\langle q_{1,1}, q_{1,2}\rangle\,\langle q_{2,1}, q_{2,2}\rangle\ldots$ in $\mathcal{A}_1 \cup \mathcal{A}_2$ we have:

$$\varsigma\big|_Q \;\models\; \mathrm{Fin}\,(U_w \times Q_2) \;\wedge\; \mathrm{Inf}\,(L_w \times Q_2)$$

We conclude that $\varsigma\big|_Q$ is an accepting run for the word w in $\mathcal{A}_1 \cup \mathcal{A}_2$. □

3.3 Implementation and Experiments

We have implemented the HP-algorithm in a tool we refer to as `ltl2gfg`. Based on `ltl2gfg`, we have additionally implemented the GFG-based quantitative analysis of MDPs in `PRISM`. After a brief overview of `ltl2gfg`, we report on our experiments and comparison with the determinization approach of `ltl2dstar`. Our implementation and the logs of the experiments are available at [Mül18].

3.3.1 LTL2GFG

Given an LTL formula φ, our implementation `ltl2gfg` constructs a symbolic, BDD-based representation of a GFG-NPA for φ. It first converts φ into an (explicitly represented) NBA $\mathcal{B}$. In our experiments, we use `ltl2ba` v1.1 [GO01] for this task. To facilitate an efficient symbolic representation of the various subsets used in the HP-algorithm, $\mathcal{B}$ is then converted to a symbolic representation, using a unary encoding of the $|Q| = n$ states of $\mathcal{B}$, i.e., using one Boolean variable q_i per state.[1] The state space of the GFG-automaton $\mathcal{A}$, i.e., the n pairs (A_i, B_i) is likewise encoded by n^2 Boolean variables $a_{i,j}$ and $b_{i,j}$, i.e., $a_{i,j}$ is true iff NBA state $q_j \in A_i$ and $b_{i,j}$ is true

[1] As the states and transition relation of the GFG automaton $\mathcal{A}$ have to encode *sets* of NBA states, such a unary encoding is the most straightforward and natural choice. A similar situation exists when considering symbolic determinization (e.g., [MS08]). As it is crucial for performance to keep the number of BDD variables low, this also negates the potential benefits of symbolic LTL-to-NBA translations (e.g., [CGH97; RV10]), as their use would require a very large number of BDD variables to allow the encoding of sets of NBA states.

iff $q_j \in B_i$ for $1 \leq i, j \leq n$. To allow the encoding of the transition relations of $\mathcal{A}$ and $\mathcal{B}$, each state variable has a primed copy, i.e., q'_i, $a'_{i,j}$ and $b'_{i,j}$ and each of the k atomic proposition in φ is represented by a Boolean variable l_i. Using these Boolean variables, the symbolic encoding of the transition relation of the NBA $\mathcal{B}$ via a BDD-based switching function is straightforwardly obtained by relating the l, q and q' variables according to the valid choices of successor state. Likewise, the definitions for the transition relation and acceptance condition of $\mathcal{A}$ in the HP-algorithm directly induce the Boolean BDD operations necessary for building the corresponding switching functions. For the performance of a BDD-based symbolic representation, the order of the variables is crucial. As is standard for BDD-based automata encodings, the state variables and their copies are always kept adjacent. The standard variable ordering used by `ltl2gfg` is then an interleaving of the $a_{i,j}$ and $b_{i,j}$ variables with the q_j variables, i.e.,

$$l_1 < \ldots < l_k \ < \ q_1 < \ldots \ < \ q_j < a_{1,j} < b_{1,j} < a_{2,j} < b_{2,j} < \ldots < q_{j+1} < \ldots .$$

This ordering ensures that the related parts of the transition relation tend to be in proximity and has fared best in our initial experiments for finding a sensible ordering. As detailed below, we also optionally used dynamic reordering techniques based on sifting [Rud93] for the variable order to heuristically reduce the BDD size of the symbolic encoding. `ltl2gfg` uses the `JINC` C++ BDD library [Oss10] for the symbolic representation.

Experimental results for the HP-algorithm We report here on a number of experiments with `ltl2gfg` using the benchmark formulas used in the evaluation of `ltl2dstar` in [KB06; KB07], i.e., 39 LTL formulas from the literature [EH00; SB00] and 55 pattern formulas [DAC99] that represent common specification patterns. We denote the generated automata equivalent to formula φ by $\mathcal{A}_\varphi$. All our experiments were carried out on a computer with 2 Intel E5-2680 8-core CPUs at 2.70 GHz with 384GB of RAM running Linux and with a memory limit of 10 GB and a timeout of 30 minutes for each formula.

For every automaton $\mathcal{A}_\varphi$, we report on the number of BDD nodes in the encoding of the transition function, as this is the most crucial aspect. To allow a fair comparison with the explicit determinization in `ltl2dstar`, we consider symbolic encodings of the DRA $\mathcal{A}_\varphi$ obtained from `ltl2dstar` 0.5.1, using the same LTL-to-NBA translator, i.e., `ltl2ba`, as used by `ltl2gfg`. This encoding uses $\lceil \log_2 n \rceil$ Boolean variables to straightforwardly encode the n state indices in $\mathcal{A}_\varphi$, which is the same encoding employed in `PRISM` for its DRA-based approach to LTL model checking.

Table 3.1 and Table 3.2 present statistics for the construction of DRA with `ltl2dstar` and GFG-NPA/NRA[2] with `ltl2gfg` for the benchmark formulas. Fig-

[2]The GFG automata obtained directly using the HP-algorithm have parity acceptance, while the automata obtained by the "union" of multiple GFG automata have Rabin acceptance, as the union construction does not preserve the special structure of parity acceptance when it considers the original NPA as NRA in the construction.

		$\mathcal{A}_\varphi$ with constr. time			
	aborted	$<1s$	$<10s$	$<1m$	$<30m$
`ltl2dstar` std.	0	90	90	92	94
no opt.	0	89	90	90	94
`ltl2gfg` std.	43	36	45	48	51
std., dynamic	45	23	36	45	49
loose, dynamic	36	35	48	55	58
lo., union, dyn.	36	45	54	55	58
lo., iterative	22	72	72	72	72
lo., it., un., dyn.	20	67	70	72	74

Table 3.1: Statistics for the automata $\mathcal{A}_\varphi$ constructed for the 94 benchmark formulas. Number of $\mathcal{A}_\varphi$ constructed within a given time frame.

		$\mathcal{A}_\varphi$ with BDD size					
	aborted	<10	$<10^2$	$<10^3$	$<10^4$	$<10^5$	$\geq 10^5$
`ltl2dstar` std.	0	4	65	87	90	91	3
no opt.	0	3	48	78	89	90	4
`ltl2gfg` std.	43	3	6	19	26	36	15
std., dynamic	45	5	8	19	36	39	10
loose, dynamic	34	5	14	31	47	55	3
lo., union, dyn.	36	4	13	35	53	55	3
lo., iterative	22	3	19	39	60	72	0
lo., it., un., dyn.	20	4	32	63	70	74	0

Table 3.2: Statistics for the automata $\mathcal{A}_\varphi$ constructed for the 94 benchmark formulas. Number of $\mathcal{A}_\varphi$ constructed within a given range of BDD sizes.

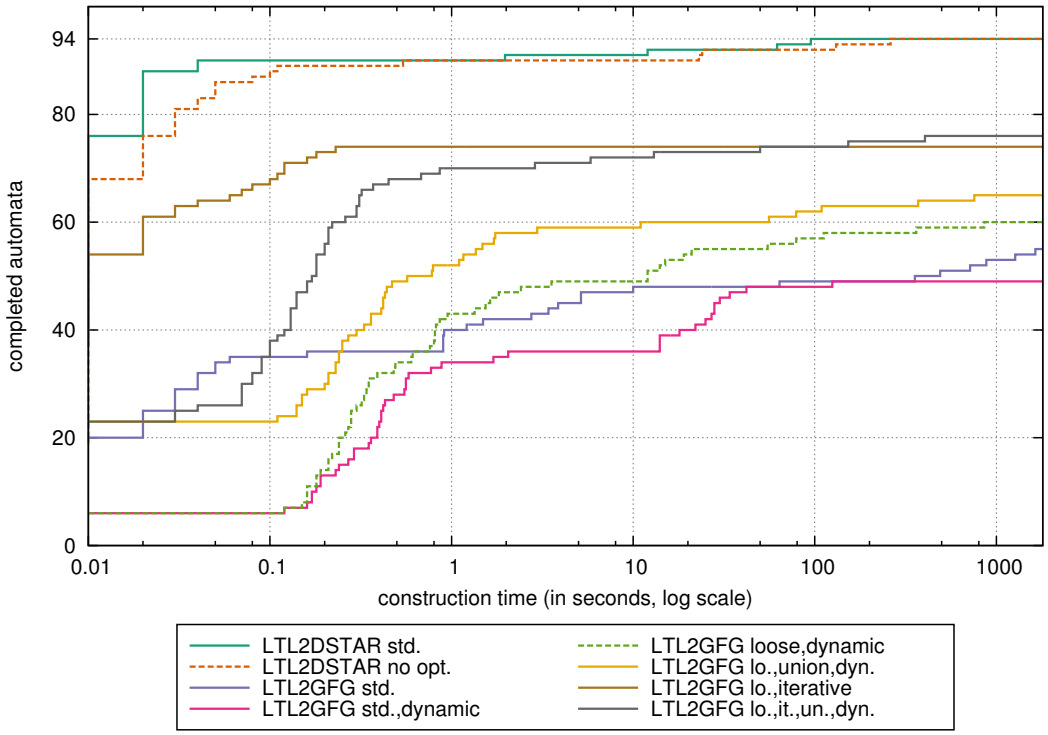


Figure 3.5: Number of automata for the 94 benchmark formulas that were constructed in the given amount of time (timeout at 1800 seconds/30 minutes), cf. Table 3.1. The construction times are aggregated at a resolution of 2 decimal places and presented in logarithmic scale to provide detail in the sub-second range.

ure 3.5 depicts the results on the construction times presented in Table 3.1 in graphical form and in more detail. The `ltl2dstar` results are given once with standard settings and for a variant where all optimizations are disabled, i.e., with purely Safra's construction. For `ltl2gfg`, we start with the pure HP-algorithm and consider variants with the "loose" transition definition, the union construction, and with dynamic reordering of the variable order. We also give statistics for the iterative approach, where `ltl2gfg` constructs the partial automata $\mathcal{A}^m$ until it can be shown (via solving a Rabin game [PP06]) that the automaton is GFG.

`ltl2dstar` (with optimizations) constructed most of the automata in a few seconds, the most difficult was constructed in 863s and had 1.1 million BDD nodes if dynamic reordering was enabled, and 254s and 1.2 million BDD nodes, if dynamic reordering was not enabled. Apart from the most difficult automata, the BDD sizes range in the hundreds and thousands. For all the `ltl2gfg` variants, a significant fraction of automata could not be constructed in the time and memory limits, around 40% for the standard HP-algorithm, and dropping to around 20% for the best variant. The loose variant by itself had a mixed effect, but in conjunction with dynamic reordering was generally beneficial. The union construction was very beneficial for

	with n NBA states											
	2	3	4	5	6	7	8	9	10	11	12	>12
number of φ	13	17	13	9	8	3	3	1	4	2	4	11
number of φ, $M < n$	11	17	13	8	8	1	2	1	0	0	1	2
number of φ, $M = 1$	11	8	5	4	2	1	1	0	0	0	1	2
number of φ, $M = 2$	2	9	8	4	6	0	1	1	0	0	0	0
number of φ, GFG check aborted	0	0	0	1	0	2	1	0	4	2	3	9

Table 3.3: Results of the iterative approach in `ltl2gfg`, for the loose variant. M is the minimal value $m \leq n$ for which the partial NPA $\mathcal{A}^m$ could be shown to be GFG.

the disjunctive formulas. For example, the automata for $\Box\Diamond a \rightarrow \Box\Diamond b$ could not be constructed in the time limits with the standard HP-algorithm but could be handled using the union construction. The iterative approach was successful as well in obtaining smaller automata, which is explained by the fact that for a large number of formulas it could be shown that the partial automata $\mathcal{A}^1$ or $\mathcal{A}^2$ were already GFG, as detailed in Table 3.3. For the iterative approach we were mostly focused on experimental data for the minimal value m for which $\mathcal{A}^m$ becomes GFG and the effect on the BDD size. Clearly, these values do not depend on our particular choice of GFG check or its implementation. Interestingly, even though our choice of GFG check is theoretically expensive, it turned out to be quite effective in practice, as can be seen in the corresponding rows of Table 3.1.

Table 3.4 lists the size of the constructed automata in terms of the number of reachable states in more detail for a selection of the benchmark formulas representing simple, typical patterns relevant for probabilistic model checking. We consider here the standard variant of the HP-algorithm [HP06] and combinations of the loose, the union and the iterative variant. We contrast this with the results of `ltl2dstar` in the default variant. Missing entries correspond to timeouts during the generation. Table 3.5 lists the corresponding size of the automata in terms of BDD nodes used for encoding the transition function of the automata. The tables were generated with dynamic reordering of the variable order activated.

In Table 3.5, we see as well one of the formulas where the BDD size of the GFG automaton was (marginally) better than the BDD size of the DRA obtained by `ltl2dstar`. For $\Box a$ the BDD for the transition function consists of 8 nodes in `ltl2gfg`, while the BDD of `ltl2dstar` consists of 10 nodes.

At the end, despite the various approaches implemented in `ltl2gfg`, there were only 6 formulas with relatively small automata where the BDD size of the smallest GFG automaton was smaller than that of the DRA obtained from `ltl2dstar` (172 nodes instead of 229 nodes, 219 instead of 347, and the other 4 automata differing by 1 or 2 at a size of less than 20 nodes). For all of our benchmark formulas [KB06; KB07; EH00; SB00; DAC99] none of the GFG automata had a smaller number of

Formula	`ltl2gfg`					`ltl2dstar`
	standard	loose	loose, union	loose, iterative	loose, iterative, union	
$true$	3	3	3	3	3	2
$false$	2	2	2	2	2	1
a	6	8	8	4	4	3
$\Box a$	3	3	3	3	3	3
$\Diamond a$	16	46	46	6	6	2
$\Diamond\Box a \to \Box\Diamond b$	–	$3.3 \cdot 10^9$	5329	6482	81	2
$\Diamond\Box a \to \Box\Diamond b$	–	$2.2 \cdot 10^9$	3358	–	414	4
$a \,\mathcal{U}\, b$	16	46	46	6	6	3
$a \,\mathcal{U}\, (b \,\mathcal{U}\, c)$	224	7734	7734	12	12	4
$\Box(a \to \Diamond b)$	33	73	73	9	9	4
$\Box(a \to \Box b)$	33	73	73	9	9	4
$(\Diamond a) \to (b \,\mathcal{U}\, a)$	135	5837	54	89	10	5
$\Box a \to (\Diamond(b \wedge \bigcirc\Diamond c))$	–	$8.3 \cdot 10^{11}$	$8.3 \cdot 10^{11}$	243	243	2
$(\Diamond(a \wedge \bigcirc\Diamond b)) \to (\neg a \,\mathcal{U}\, c)$	–	$2.2 \cdot 10^9$	3358	4295	54	6
$(a \,\mathcal{U}\, b) \vee \Box a$	434	11952	11952	460	460	4
$(\Box\neg a) \vee (\Diamond(b \wedge \Diamond a))$	–	$5.3 \cdot 10^6$	7750	341	16	4

Table 3.4: Detailed statistics for example formulas: number of reachable states.

Formula	ltl2gfg					ltl2dstar
	standard	loose	loose, union	loose, iterative	loose, iterative, union	
true	7	7	7	7	7	3
false	7	7	7	7	7	1
a	59	46	46	17	17	11
$\Box a$	8	8	8	8	8	10
$\Diamond a$	123	85	85	22	22	6
$\Diamond\Box a \rightarrow \Box\Diamond b$	–	69669	147	2383	51	6
$\Diamond\Box a \rightarrow \Box\Diamond b$	–	12803	149	–	107	8
$a \,\mathcal{U}\, b$	132	102	102	23	23	14
$a \,\mathcal{U}\, (b \,\mathcal{U}\, c)$	5165	1642	1642	62	62	21
$\Box(a \rightarrow \Diamond b)$	182	97	97	31	31	12
$\Box(a \rightarrow \Box b)$	99	71	71	18	18	19
$(\Diamond a) \rightarrow (b \,\mathcal{U}\, a)$	65735	2806	119	484	33	22
$\Box a \rightarrow (\Diamond(b \wedge \bigcirc\Diamond c))$	–	93155	93155	646	646	190
$(\Diamond(a \wedge \bigcirc\Diamond b)) \rightarrow (\neg a \,\mathcal{U}\, c)$	–	41729	198	1019	48	29
$(a \,\mathcal{U}\, b) \vee \Box a$	4186	623	623	236	236	17
$(\Box\neg a) \vee (\Diamond(b \wedge \Diamond a))$	–	24874	1499	1554	68	12

Table 3.5: Detailed statistics for example formulas: size of the transition function BDD.

states than the DRA generated by `ltl2dstar`. In particular, the automata obtained without the iterative approach often had millions and more states.

3.3.2 Implementation and experiments in PRISM

Despite the negative results of our experiments in Section 3.3.1, we investigated the use of good-for-games automata in the context of probabilistic model checking. It could be the case that particularities of the symbolic encoding or of the automaton's structure turn out to be beneficial in this setting.

We extended the MTBDD-based, symbolic engine of `PRISM` 4.1 with an implementation of our algorithm for computing $\Pr^{\max}_{\mathcal{M}}(\varphi)$ using GFG automata for φ (and $\Pr^{\min}_{\mathcal{M}}(\varphi)$ using a GFG automaton for $\neg\varphi$). We import the BDD of $\mathcal{A}$ generated with `ltl2gfg` into `PRISM` and perform the product with $\mathcal{M}$ and analysis in $\mathcal{M} \otimes \mathcal{A}$ symbolically.

We compare this approach with the standard approach of `PRISM`, where an explicit DRA is constructed with an integrated version of `ltl2dstar`, which is then symbolically encoded as described before. Additionally, we evaluate our case studies on `Delag`, our tool for generating deterministic Emerson-Lei automata (see Chapter 5). The analysis is then carried out symbolically by the `MTBDD` engine for every approach, i.e., where the matrix and the value vector are stored via MTBDDs. Using the `hybrid` engine, which stores the value vector explicitly, would be obstructive for the GFG approach, since the automata have a large state number, and thus the product.

If `PRISM` normally handles a formula via a specialized algorithm for simple path formulas that does not need an automata product construction, we forced the use of the general, automata-based approach. We have carried out our experiments with the different variants for the generation of GFG automata of `ltl2gfg`. As before, we impose a 30 minute time and 10GB memory limit.

`PRISM` case study: IEEE 802.11. For our first experiment, we use a `PRISM` model [KNS02] from the `PRISM` benchmark suite for parts of the WLAN carrier-sense protocol of IEEE 802.11. For details on the model we refer to `http://www.prismmodelchecker.org/casestudies/wlan.php`

It models a two-way handshake mechanism of the IEEE 802.11 (WLAN) medium access control scheme with two senders that compete for the medium. As messages get corrupted when both senders send at the same time (called a collision), a probabilistic back-off mechanism is employed to reduce the likelihood of collisions. The back-off procedure is the key feature of the protocol, which is started if an error occurred or the sender wants to send a new message after sending a message. The back-off procedure consists of waiting a randomized amount of time while the channel has to be free. It ends with retrying to send a message. To define the maximal amount of waiting time in the back-off procedure, the model is parametrized by the parameter `MAX_BACKOFF`. Here, we consider the values $\texttt{MAX\_BACKOFF} \in \{1, \ldots, 6\}$. Since stations cannot listen to their own transmissions, after having started to transmit a message, they cannot determine for a short amount of time whether another station has started

to send at the same moment, called the *vulnerable* section. To reduce the likelihood of these collisions, a station has to check that the channel is free for a fixed time period. This happens in state `Wait_Difs`. In the model, a collision counter is used to record the number of collisions for use in the formulas. For our experiments, we set the maximum number of collisions that can be counted to 4.

For this benchmark we consider the following LTL path properties:

- $\varphi_1 = \Diamond(\texttt{correct}_1 \wedge \texttt{correct}_2)$
  "Eventually station 1 and station 2 have sent their message correctly."

- $\varphi_2 = \neg\Diamond\Box(\texttt{Backoff}_1 \vee \texttt{Wait\_Difs}_1 \vee \texttt{Wait\_ack}_1)$
  "Station 1 never gets stuck in the back-off procedure or during one of the waiting procedures."

- $\varphi_3 = (\Box\Diamond\texttt{Backoff}_1) \rightarrow (\Box\Diamond\texttt{vuln}_1)$
  "If station 1 backs off infinitely often, it is also infinitely often in its vulnerable section."

- $\varphi_4 = \Box(\texttt{col} \rightarrow \Diamond\texttt{msg\_1\_send})$
  "If a collision occurred on the channel, then station 1 will nevertheless send its message correctly at some point in the future."

- $\varphi_5 = \Box(\texttt{col} \rightarrow \Diamond(\texttt{msg\_1\_send} \wedge \texttt{\#col} < 2))$
  "Like φ_4, but with the additional constraint of no more additional collisions."

- $\varphi_6 = \Diamond(\texttt{correct}_1 \wedge \texttt{correct}_2) \quad \wedge \quad \Box(\texttt{\# col} < 2)$
  "Eventually both stations have sent their messages and the number of collisions never exceeds 1."

We also evaluated the properties we use in Section 5.4.2 (Benchmark for Emerson-Lei acceptance) for the model with `MAX_BACKOFF` = 3, but the GFG approach performed so poorly, that an evaluation based on the properties in Section 5.4.2 does not make sense. Only the calculation for two properties could be performed within the time limit of 30 minutes and memory bound of 10 GB at least for some GFG variants: $\Pr^{\min}(\varphi_1)$ and $\Pr^{\min}(\varphi_2)$. For $\Pr^{\min}(\varphi_1)$, 9 of the 16 variants were able to complete the calculation, mostly variants that contained the iterative computation. The iterative variant with dynamic reordering was the fastest (1120.7 seconds). The setting `PRISM` with `Delag` (our tool generating Emerson-Lei automata, see Chapter 5) was able to complete the calculation for $\Pr^{\min}(\varphi_1)$ within 10.2 seconds.

	PRISM ltl2gfg		PRISM standard	
	t_{MC}	$\lvert\mathcal{M} \otimes \mathcal{A}\rvert$	t_{MC}	$\lvert\mathcal{M} \otimes \mathcal{A}\rvert$
$\Pr^{\min}(\varphi_1)$	15.7 s	20,937	6.5 s	20,961
$\Pr^{\min}(\varphi_2)$	22.4 s	24,812	0.5 s	22,668
$\Pr^{\min}(\varphi_3)$	45.9 s	34,191	1.6 s	25,151
$\Pr^{\min}(\varphi_4)$	27.1 s	23,123	2.4 s	21,382
$\Pr^{\min}(\varphi_5)$	147.5 s	22,502	100.8 s	21,005
$\Pr^{\min}(\varphi_6)$	100.8 s	21,348	44.0 s	21,085

Table 3.6: Results for model checking WLAN3. For every approach the overall model checking time $t_{\mathcal{M}}$, and the BDD size of the product $\lvert\mathcal{M} \otimes \mathcal{A}\rvert$ are listed.

For $\Pr^{\min}(\varphi_2)$ every variant was able to complete the calculation within the time bound. The variant union with dynamic reordering was here the fastest: 425.2 seconds. In comparison, PRISM with Delag took 516.2 seconds, which was the worst for Delag under all properties listed in Section 5.4.2.

In our tables, we will refer by WLANn to the case-study MDP with maximum back-off value MAX_BACKOFF, for n from 1 to 6. Table 3.6 lists the time spent for calculating $\Pr^{\min}_{\mathcal{M}}(\varphi)$ for each of the six LTL formulas φ and the WLAN3 MDP. For this table, we list the results using the optimal variant of ltl2gfg for each formula. In all cases in this table, the time spent generating the GFG automata was less than one second. This allows a fair comparison against the standard PRISM approach using DRA. Interestingly, the BDD sizes listed in Table 3.6 for the number of MTBDD nodes used to encode the transition matrix of the product $\mathcal{M} \otimes \mathcal{A}$ is roughly similar between the standard approach based on DRA and the best ltl2gfg approach. However, the time then spent for calculating the probabilities in the product are vastly higher for the GFG approach than for the DRA approach.

Table 3.7 lists the time spent in model checking the different MDPs and formulas in PRISM using the standard approach. Table 3.8 lists the corresponding values for the approach using ltl2gfg in the loose variant, which also employed the iterative and union approach. This variant generally behaved well in these experiments. We did not use dynamic variable reordering here, since it turned out to be not beneficial. We list as well the time spent for constructing the GFG NRA for $\neg\varphi_i$. The formulas are negated because we are interested in the minimal probabilities.

To provide an overview of the behavior of the different variants of ltl2gfg in the context of PRISM, Table 3.9 compares the running time of some variants against the baseline of the DRA-based standard approach. We consider the 36 combinations of WLAN1 to WLAN6 and φ_1 to φ_6. The table lists the number of cases where a timeout occurred and where time spent using the GFG approach exceeded the standard approach only by a given factor. We refer to the baseline time spent using the standard approach in PRISM as t_{STD} and to the time spent using the GFG approach (when there was no timeout) as t_{GFG}. For example, if we consider the loose variant with active iterative approach, in 11 of the 36 cases the running time of PRISM

$\Pr^{\min}(\cdot)$	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6
WLAN1	0.8 s	0.1 s	0.4 s	0.4 s	9.2 s	5.7 s
WLAN2	2.1 s	0.3 s	0.8 s	1.1 s	27.5 s	18.5 s
WLAN3	6.5 s	0.5 s	1.6 s	2.4 s	100.8 s	58.8 s
WLAN4	16.6 s	0.7 s	4.6 s	6.3 s	331.5 s	186.0 s
WLAN5	58.2 s	1.1 s	15.3 s	19.7 s	1058.5 s	443.1 s
WLAN6	170.3 s	2.1 s	79.4 s	97.6 s	–	1226.4 s

Table 3.7: Time consumption for model checking with standard PRISM.

$\Pr^{\min}(\cdot)$	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6
Constructing GFG $\mathcal{A}_{\neg\varphi_i}$	0.3 s	0.3 s	0.5 s	0.3 s	0.3 s	0.4 s
WLAN1	2.6 s	4.0 s	9.6 s	4.7 s	17.0 s	13.2 s
WLAN2	6.1 s	9.8 s	19.9 s	10.8 s	52.5 s	37.9 s
WLAN3	18.1 s	26.8 s	53.9 s	29.6 s	162.3 s	116.8 s
WLAN4	66.1 s	87.6 s	189.5 s	113.1 s	513.7 s	374.4 s
WLAN5	222.2 s	322.4 s	660.7 s	365.0 s	–	1139.8 s
WLAN6	765.4 s	1101.9 s	–	1297.4 s	–	–

Table 3.8: Time consumption for PRISM and ltl2gfg (variant loose, iterative and union).

	std.	loose	loose dyn.	loose union	loose it.	loose it. dyn.	loose it. union
$t_{\text{GFG}} < 3 \cdot t_{\text{STD}}$	7	8	7	7	7	7	9
$t_{\text{GFG}} < 7 \cdot t_{\text{STD}}$	11	11	11	15	16	13	15
$t_{\text{GFG}} < 20 \cdot t_{\text{STD}}$	17	17	17	21	22	21	21
$t_{\text{GFG}} < 250 \cdot t_{\text{STD}}$	29	27	29	30	32	28	30
$t_{\text{GFG}} \leq 30m$	31	28	30	32	33	32	32
Aborted	5	8	6	4	3	4	4

Table 3.9: Results for model checking IEEE 802.11 with PRISM and different variants of ltl2gfg.

with the GFG approach was within the time spent by the standard PRISM approach multiplied by the factor 7. As can be seen, the loose variant with union and the iterative approach fared well in general. Interestingly, the automata generated with active dynamic reordering in some cases fared significantly worse than those using the initial variable ordering. As PRISM does not support a reordering of the variables, the BDD representation of the GFG automaton is optimized by the dynamic reordering in ltl2gfg for the stand-alone representation. Clearly, this variable ordering may however not be optimal for the product with the MDP. Likewise, the dynamic variable reordering sometimes slowed down the GFG check in the iterative approach.

As it was to be expected given our results on the automata construction, the GFG-based analysis did not improve on the standard approach. Even using the optimal variant of ltl2gfg for each formula, ignoring the automata construction times, and for cases where the product $\mathcal{M} \otimes \mathcal{A}$ had a comparable BDD size for the GFG- and DRA-based approach, the model checking using the GFG automata took significantly longer.

PRISM case study: Dining philosophers As a second benchmark, we consider the well-known problem of the dining philosophers. Lehmann and Rabin [LR81] presented a probabilistic solution to this problem, which was analyzed by Lynch, Saias and Segala [LSS94] for its timing behavior. Our model from the PRISM benchmark suite, see http://www.prismmodelchecker.org/casestudies/phil_lss.php, consists of three philosophers, who sit around a table with a fork on the right side of each philosopher. Each philosopher either eats or thinks. To eat, the philosopher has to grab both the fork to the left and to the right. The order in which each philosopher wants to grab the forks is determined probabilistically. The PRISM model introduces an additional constant K, which restricts the maximal number of transitions that a philosopher can be forced to wait if he is currently attempting to grab a fork. For our benchmark we chose $K = 4$.

We considered the following LTL properties:

- $\varphi_1 = \square(p_1 = \texttt{wait} \rightarrow \lozenge p_1 = \texttt{eat})$
  "If philosopher 1 is hungry, he will eat eventually"

- $\varphi_2 = \square\lozenge\,(p_1 = \texttt{wait} \vee p_2 = \texttt{wait} \vee p_3 = \texttt{wait})$
  $\rightarrow \square\lozenge\,(p_1 = \texttt{eat} \vee p_2 = \texttt{eat} \vee p_3 = \texttt{eat})$
  "If one philosopher is hungry infinitely often, one philosopher eats infinitely often"

- $\varphi_3 = \square\neg\big((p_1 = \texttt{eat} \wedge p_2 = \texttt{eat}) \vee (p_1 = \texttt{eat} \wedge p_3 = \texttt{eat}) \vee (p_2 = \texttt{eat} \wedge p_3 = \texttt{eat})$
  "Only one of the philosophers can eat at the same time"

- $\varphi_4 = \lozenge(p_1 = \texttt{eat} \vee p_2 = \texttt{eat} \vee p_3 = \texttt{eat})$
  "Eventually at least one philosopher eats"

	PRISM ltl2gfg			PRISM standard		
	t_{GFG}	t_{MC}	$\lvert\mathcal{M}\otimes\mathcal{A}\rvert$	t_{DRA}	t_{MC}	$\lvert\mathcal{M}\otimes\mathcal{A}\rvert$
$\Pr^{\max}_{\mathcal{M}}(\varphi_1)$	0.3 s	2.3 s	21,155	$<$0.1 s	1.1 s	18,865
$\Pr^{\min}_{\mathcal{M}}(\varphi_1)$	0.3 s	4.2 s	18,946	$<$0.1 s	0.7 s	18,336
$\Pr^{\min}_{\mathcal{M}}(\varphi_2)$	1.5 s	12.9 s	78,667	$<$0.1 s	1.2 s	54,460
$\Pr^{\min}_{\mathcal{M}}(\varphi_3)$	0.3 s	0.7 s	13,222	$<$0.1 s	$<$0.1 s	13,204
$\Pr^{\max}_{\mathcal{M}}(\varphi_4)$	0.3 s	2.4 s	16,188	$<$0.1 s	1.6 s	22,692
$\Pr^{\min}_{\mathcal{M}}(\varphi_5)$	0.3 s	2.5 s	35,904	$<$0.1 s	0.2 s	22,932
$\Pr^{\min}_{\mathcal{M}}(\varphi_6)$	0.3 s	5.3 s	40,156	$<$0.1 s	0.7 s	36,840
φ_7	3.7 s	8.3 s	92,141	$<$0.1 s	4.8 s	57,171

Table 3.10: Results for model checking dining philosophers. The time for probabilistic model checking (t_{MC}) includes the time for building the automaton (t_{GFG} for building the GFG automaton, t_{DRA} for building the DRA).

- $\varphi_5 = \Box\Diamond(p_1 = \texttt{think} \vee p_1 = \texttt{wait} \vee p_1 = \texttt{eat} \vee p_2 = \texttt{think} \vee p_2 = \texttt{wait} \vee p_2 = \texttt{eat} \vee p_3 = \texttt{think} \vee p_3 = \texttt{wait} \vee p_3 = \texttt{eat})$
  "Infinitely often one philosopher thinks, waits or eats"

- $\varphi_6 = \Box\big((p_1 = \texttt{wait} \vee p_2 = \texttt{wait} \vee p_3 = \texttt{wait}) \rightarrow (\Diamond(p_1 = \texttt{eat} \vee p_2 = \texttt{eat} \vee p_3 = \texttt{eat}))\big)$
  "If some philosopher wants to eat, one philosopher will eat eventually"

- $\varphi_7 = \min_{s\in S_0}\big(\Pr^{\min}_{\mathcal{M},s}(\Diamond^{\leq 30}(p_1 = \texttt{took} \vee p_2 = \texttt{took} \vee p_3 = \texttt{took}))\big)$
  where
  $S_0 = \{s \in S : s \models p_1 = \texttt{try} \vee p_2 = \texttt{try} \vee p_3 = \texttt{try}\}$ and
  $\Diamond^{\leq n}\varphi = \varphi \vee (\bigcirc\,\varphi) \vee (\bigcirc\,\bigcirc\,\varphi) \vee \ldots \vee (\underbrace{\bigcirc \ldots \bigcirc}_{n \text{ times}}\varphi)$.
  "If one philosopher tries to eat, one philosopher will take two forks ($p_i = \texttt{took}$) within 30 transitions"

For φ_7, we rely on PRISM's "filter" mechanism, where the inner $\Pr^{\min}$ is evaluated for all relevant states in one go, subsequently applying the outer min operator to the results.

Table 3.10 lists statistics for our benchmark experiments when using the GFG approach in PRISM and when using the standard, deterministic automata based approach. For the GFG approach, we list results for the best of the different GFG variants for each formula, determined by the overall time spent for generating the automaton and the subsequent model checking. The first two columns for ltl2gfg and the PRISM standard approach, respectively, list the time spent for constructing the automaton and for model checking, where the model checking time includes the

time for automata construction. The remaining columns list the number of MTBDD nodes used for representing the product of the model and the automaton.

As in the previous case study, model checking using the GFG automata took longer than the standard approach. For property φ_7, all variants with the iterative optimization but without dynamic reordering finished with a running time between 8 and 9 seconds, while all iterative variants with dynamic reordering finished with a running time between 618 and 633 seconds. All non-iterative variants exceeded the time limit during the automaton construction for φ_7.

	dyn.	loose dyn.	loose it.	it. un. dyn.	loose it. un.	loose	un.	loose un. dyn.
$t_{\text{GFG}} \leqslant 2 \cdot t_{\text{std}}$	1	0	2	0	2	0	0	0
$t_{\text{GFG}} \leqslant 5 \cdot t_{\text{std}}$	2	2	3	1	3	2	2	2
$t_{\text{GFG}} \leqslant 10 \cdot t_{\text{std}}$	3	2	3	3	3	2	4	2
$t_{\text{GFG}} \leqslant 25 \cdot t_{\text{std}}$	7	5	7	6	7	5	7	5
$t_{\text{GFG}} \leqslant 50 \cdot t_{\text{std}}$	7	7	8	6	8	7	7	7
$t_{\text{GFG}} \leqslant 30m$	7	7	8	8	8	7	7	7
Aborted	1	1	0	0	0	1	1	1

Table 3.11: Results for model checking dining philosophers with PRISM and different variants of ltl2gfg.

Analogously to Table 3.9, Table 3.11 provides details for the model checking time for several variants of ltl2gfg against the standard approach. For the dining philosophers, the iterative and union approach without dynamic reordering behaved well in comparison to other ltl2gfg variants. On the other hand, when the union optimization or the iterative approach is disabled, the time consumption increases.

As a whole, the model checking results confirm the result of the case study for IEEE 802.11. We were not able to outperform the standard PRISM approach, even if we use the optimal variant of ltl2gfg for each property.

3.4 Conclusion

We have demonstrated that good-for-games automata can be used instead of deterministic automata for the quantitative analysis of MDPs against ω-regular properties. Our approach has the same asymptotic worst-case time complexity as the standard approach with deterministic automata. We have implemented the HP-algorithm to evaluate whether GFG automata offer advantages over deterministic automata in practice. We have performed extensive experiments for the generation of GFG automata from NBA for given LTL formulas and for probabilistic model checking, considering several variants and heuristics that can lead to dramatic improvements in the size of the constructed automata. However, the GFG automata were often bigger

than DRA obtained by the implementation of Safra's algorithms in `ltl2dstar`. This was evident not only in the number of states in the generated automata, which is not as crucial for a symbolic encoding, but also in the size of the symbolic BDD-based representations, in contrast to the expectation that the HP-algorithm could potentially construct GFG automata with a more efficient symbolic encoding. Our experiments with the GFG-based approach implemented in probabilistic model checker PRISM then likewise resulted in a higher time and memory consumption compared to the deterministic automaton approach.

However, it is still too early to discard the concept of GFG automata for practical purposes as our negative results in the experiments might be an artifact of the particular translation algorithm. Our negative empirical results might be an artifact of the HP-algorithm, which is – to the best of our knowledge – the only implemented algorithm for the generation of GFG automata that are not deterministic. Therefore, it would be interesting to consider alternative constructions that lead to succinct GFG automata for an explicit or symbolic representation. The recent results on GFG automata that have fewer states than equivalent deterministic automata [KS15] might provide ideas in this direction. It would also be interesting to consider constructions that attempt to exploit the potential benefits in succinctness offered by targeting Rabin and Streett acceptance conditions directly instead of using parity acceptance. A short while ago, Kuperberg and Majumdar [KM18] designed a new generation algorithm with non-deterministic co-Büchi automata as starting point. Additionally, they sketched an adaption of Safra's determinization for achieving the GFG property, but an evaluation of the proposed algorithms remains open.

Likewise, there is an interest in discovering alternative automata types that rely on weaker conditions than the GFG property but can nevertheless be employed for probabilistic model checking or other settings. One example of such a property would be GFG-in-the-limit, but the substantial problem of constructing such succinct automata remains.

4 Unambiguous Büchi automata

Unambiguity is a widely studied generalization of determinism with many important applications in automata-theoretic approaches, see, e.g., [Col12; Col15]. A non-deterministic automaton is said to be unambiguous if each word has at most one accepting run. In this thesis we consider unambiguous Büchi automata (UBA) over infinite words. Not only are UBA as expressive as the full class of non-deterministic Büchi automata (NBA) [Arn85], they can also be exponentially more succinct than deterministic automata. For example, the language "eventually b occurs and a appears k steps before the first b" over the alphabet $\{a, b, c\}$ is recognizable by a UBA with k+1 states (see the UBA on the left of Figure 4.1), while a deterministic automaton requires at least 2^k states, regardless of the acceptance condition, as it needs to store the positions of the a's among the last k input symbols. Languages of this type arise in a number of contexts, e.g., absence of unsolicited response in a communication protocol – if a message is received, then it has been sent in the recent past. Another UBA is depicted on the right of Figure 4.1 containing a universal UBA. An ω-automaton $\mathcal{A}$ is universal, if every infinite word is accepted by $\mathcal{A}$. For the particular UBA depicted on the right side of Figure 4.1, all accepting runs for words starting with a begin in q_a, whereas all accepting runs for words starting with b begin in q_b.

The NBA for linear temporal logic (LTL) formulas obtained by applying the classical closure algorithm of [WVS83; VW86] are unambiguous. The generated automata moreover enjoy the separation property: the languages of the states are pairwise disjoint. Thus, while the generation of deterministic ω-automata from LTL formulas involves a double-exponential blow-up in the worst case, the translation of LTL formulas into separated UBA incurs only a single exponential blow-up. This fact has been observed by several authors, see, e.g., [CSS03; Mor10], and recently adapted for LTL with step parameters [Zim13; CK14]. However, separation is such a strong constraint, that even deterministic automata are not necessarily separated, although

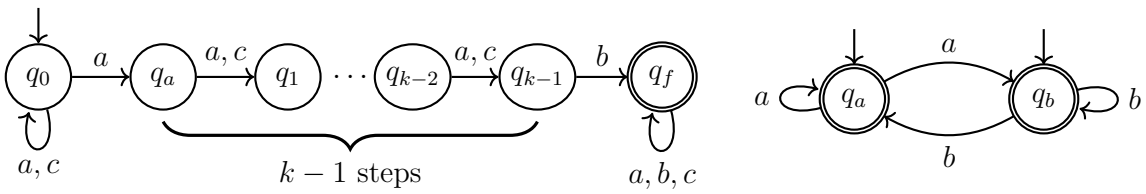


Figure 4.1: Left: UBA for "eventually b and a appears k steps before first b", right: A universal and separated UBA.

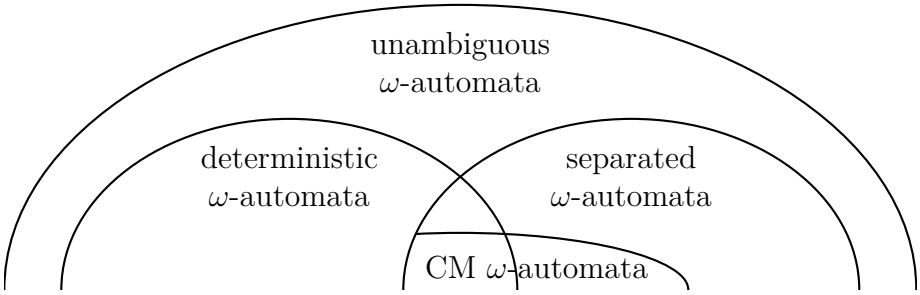


Figure 4.2: Overview of different classes of unambiguous ω-automata. CM ω-automata are Carton-Michel automata, called CUBA in [CM03]. Note that there is exactly one deterministic Carton-Michel ω-automaton.

they are always unambiguous. In [CM03], there is an even stronger notion of separated UBA, which we call CM-automata in this document. These are separated UBA that are also universal if every state is set to be initial. For a fixed alphabet, there exists exactly one deterministic CM-automaton: The trivial one-state automaton with a transition for every symbol [CM03]. The authors of [Li+16a] have developed a polynomial-time algorithm for parameter synthesis in parametric Markov chains and (non-deterministic) CM-automata. For a picture of the different classes of unambiguity, see Figure 4.2.

The nice properties of UBA make them a potentially attractive alternative to deterministic ω-automata in those applications for which general non-deterministic automata are not suitable. However reasoning about UBA is surprisingly difficult. While many decision problems for unambiguous non-deterministic finite automata (UFA) are known to be solvable in polynomial time [SH85], the complexity of several fundamental problems for unambiguous automata over infinite words is unknown. This, for instance, applies for checking universality, which is known to be in P for deterministic Büchi automata and PSPACE-complete for NBA. However, the complexity of the universality problem for UBA is a long-standing open problem. Polynomial-time solutions are only known for separated UBA and other subclasses of UBA [BL10; IL12].

The class of finitely ambiguous Büchi automata received attention lately. The property of having at most one accepting run for every word is relaxed to having only finitely many accepting runs for every word in a finitely ambiguous Büchi automaton. There exists a family of NBA, such that every equivalent finitely ambiguous Büchi automaton is at least of exponential size [LP18]. Finitely ambiguous Büchi automata can be constructed starting from an NBA similarly to [KW08] (see [LP18]), and complemented to a UBA with an exponential blowup [Rab18]. For finite words finitely ambiguous automata has been considered for a longer time. For a survey on properties, in particular the state complexity of automata, for different degrees of ambiguity, we refer to the survey paper [HSS17]. In the thesis we use the terms "unambiguous" and "unambiguity" in the more usual sense of having at most one

accepting run for every word, and do not pursue finite ambiguity further.

In the context of probabilistic model checking, UFA provide an elegant approach to compute the probability for a regular safety or cosafety property in finite-state Markov chains [BLW14]. The use of separated UBA for a single exponential-time algorithm that computes the probability for an LTL formula in a Markov chain has been presented in [CSS03]. However, separation is a rather strong condition and non-separated UBA (and even DBA) can be exponentially more succinct than separated UBA, see [BL10]. This motivates the design of algorithms that operate with general UBA rather than the subclass of separated UBA. Algorithms for the generation of (possibly non-separated) UBA from LTL formulas that are more compact than the separated UBA generated by the classical closure-algorithm have been realized in the tool `Tulip` [Len13b; Len13a] and the automata library `SPOT` [Dur13].

Contribution. The main theoretical contribution of this chapter is a polynomial-time algorithm to compute the probability measure $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{U}))$ of the set of infinite paths generated by a finite-state Markov chain $\mathcal{M}$ that satisfy an ω-regular property given by a (not necessarily separated) UBA $\mathcal{U}$. The existence of such an algorithm has previously been claimed in [BLW13b; BLW14; Len13b]. However, these previous works share a common fundamental error. Specifically they rely on the claim that if $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{U})) > 0$, then there exists a state s of the Markov chain $\mathcal{M}$ and a state q of the automaton $\mathcal{U}$ such that q accepts almost all traces emanating from s (see [BLW13b, Lemma 7.1], [BLW14, Theorem 2][1] , and [Len13b, Section 3.3.1]). While this claim is true in case that $\mathcal{U}$ is deterministic [CY95], it needs not to hold when $\mathcal{U}$ is merely unambiguous. Sections 4.1.1 and 4.1.2 give a more detailed analysis of the issue, describing precisely the nature of the errors in the proofs of [BLW13b; Len13b; BLW14]. To the best of our knowledge these errors are not easily fixable, and we thus take a substantially different approach.

Our algorithm involves a two-phase method that first analyzes the strongly connected components (SCCs) of a graph obtained from the product of $\mathcal{M}$ and $\mathcal{U}$, and then computes the value $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{U}))$ using linear equation systems. The main challenge is the treatment of the individual SCCs. For a given SCC, we have an equation system comprising a single variable and equation for each vertex (s, q), with s a state of $\mathcal{M}$ and q a state of $\mathcal{U}$. We use results in the spectral theory of non-negative matrices to argue that this equation system has a non-zero solution exactly in case that the SCC makes a non-zero contribution to $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{U}))$. In order to compute the exact value of $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{U}))$, the key idea is to introduce an additional normalization equation. To obtain the latter, we identify a pair (s, R), where s is a state of the Markov chain $\mathcal{M}$ and R a set of states of automaton $\mathcal{U}$ such that almost all paths starting in s have an accepting run in $\mathcal{U}$ when the states in R are declared to be initial. The crux of establishing a polynomial bound on the running time of our algorithm is to find such a pair (s, R) efficiently (in particular, without determinizing

[1] As the flaw is in the handling of the infinite behavior, the claim and proof of Lemma 1 in [BLW14], dealing with unambiguous automata over finite words, remain unaffected.

$\mathcal{U}$) by exploiting structural properties of unambiguous automata.

As a consequence of our main result, we obtain that the *almost universality* problem for UBA, which can be seen as the probabilistic variant of the universality problem for UBA and which asks whether a given UBA accepts almost all infinite words, is solvable in polynomial time.

The second contribution in this chapter is an implementation of the new algorithm as an extension of the model checker `PRISM`, using the automata library `SPOT` [Dur13] for the generation of UBA from LTL formulas and the `COLT` library [Hos04] for various linear algebra algorithms. We evaluate our approach using the bounded retransmission protocol case study from the PRISM benchmark suite [KNP12] as well as specific aspects of our algorithm using particularly "challenging" UBA.

4.1 Analysis of Markov chains against UBA-specifications

The task of the *probabilistic model checking problem* for a given Markov chain $\mathcal{M}$ and NBA $\mathcal{A}$ is to compute $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{A}))$. The *positive model checking problem* for $\mathcal{M}$ and $\mathcal{A}$ asks whether $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{A})) > 0$. Likewise, the *almost-sure model checking problem* for $\mathcal{M}$ and $\mathcal{A}$ denotes the task to check whether $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{A})) = 1$. While the positive and the almost-sure probabilistic model checking problems for Markov chains and NBA are both known to be PSPACE-complete [Var85; CY95], the analysis of Markov chains against UBA-specification can be carried out efficiently as stated in the following theorem:

Theorem 4.1. *Given a Markov chain $\mathcal{M}$ and a UBA $\mathcal{U}$, the value $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{U}))$ is computable in time polynomial in the sizes of $\mathcal{M}$ and $\mathcal{U}$.*

The statement of Theorem 4.1 has already been presented in [BLW13b] and restated [BLW13a; BLW14] (see also the PhD thesis [Len13b]). However, the presented algorithm to compute $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{U}))$ is flawed. This approach, rephrased for the special case where the task is to compute $\Pr(\mathcal{L}_\omega(\mathcal{U}))$ for a given positive UBA $\mathcal{U}$ (which means a UBA where $\Pr(\mathcal{L}_\omega(\mathcal{U})) > 0$) relies on the mistaken belief that there is at least one state q in $\mathcal{U}$ such that $\Pr(\mathcal{L}_\omega(\mathcal{U}[q])) = 1$. However, such states need not exist.

Outline of Section 4.1. At first, we present two counterexamples for [BLW13b] and [BLW14] as well (see also [Len13b]). The remainder of Section 4.1 is devoted to the proof of Theorem 4.1. We first assume that the Markov chain $\mathcal{M}$ generates all words according to a uniform distribution ("uniform Markov chain") and explain how to compute the value $\Pr(\mathcal{L}_\omega(\mathcal{U}))$ for a given UBA $\mathcal{U}$ in polynomial time. For this, we first address the case of strongly connected UBA (Section 4.1.3) and then lift the result to the general case (Section 4.1.4). The central idea of the algorithm relies on the observation that each positive, strongly connected UBA has "recurrent sets" of states,

called *cuts*. We exploit structural properties of unambiguous automata for the efficient construction of a cut and show how to compute the values $\Pr(\mathcal{L}_\omega(\mathcal{U}[q]))$ for the states of $\mathcal{U}$ by a linear equation system with one equation per state and one equation for the generated cut. Furthermore, positivity of a UBA $\mathcal{U}$ (i.e., $\Pr(\mathcal{L}_\omega(\mathcal{U})) > 0$) is shown to be equivalent to the existence of a positive solution of the system of linear equations for the states. Finally, we explain how to adapt these techniques to general Markov chains (Section 4.1.5).

4.1.1 Counterexample for the approach of [BLW13b]

The original proposal for using UBA for model checking of DTMCs can be found in Lemma 7.1 of [BLW13b] (p.22) which is the same as in [Len13b]. We now give a counterexample for this approach.

In [BLW13b], the Büchi automata are state-labeled, usually with an alphabet over some atomic propositions. For presentational simplicity, we use here a fixed alphabet $\{a, b, c, d\}$, corresponding to the states of the Markov chain, omitting the atomic proposition based labeling functions. The accepting condition in [BLW13b] is a state-based generalized Büchi condition Φ.

The paper [BLW13b] assumes a product graph out of a Markov chain $\mathcal{M}$ and an unambiguous generalized Büchi automaton $\mathcal{U}$. The nodes are pairs of Markov chain states and UBA states with matching labels. There is an edge between two nodes if and only if there is an edge between the two corresponding Markov chain states and an edge between the two corresponding UBA states. It defines an SCC $\mathcal{C}$ to be accepting if

(i) for every Büchi condition $\mathrm{Inf}(Z)$ occurring in the generalized Büchi condition Φ there exists a node $\langle s, q_Z \rangle$ with $q_Z \in Z$, and

(ii) for every node $\langle s, p \rangle$ and every transition $s \longrightarrow t$ in the Markov chain $\mathcal{M}$ there exists a transition $p \longrightarrow q$ in $\mathcal{U}$ such that $\langle t, q \rangle$ is contained in $\mathcal{C}$.

Example. Our counterexample will give a Markov chain and a UBA, for which $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{U})) = 1$ holds, but the product will not contain an accepting SCC according to the definition of [BLW13b]. Consider the (state-labeled) UBA $\mathcal{U}$ and the Markov chain $\mathcal{M}$ of Figure 4.3.

The UBA accepts all words of the form

$$\big((dab) + (dac)\big)^\omega$$

and consequently $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{U})) = 1$.

The product graph $\mathcal{M} \otimes \mathcal{U}$ that arises from the construction in the proof of Lemma 7.1 of [BLW13b] is depicted in Figure 4.4. It is strongly connected, but it is not accepting as condition (ii) is violated: Consider the vertex (a, q_a) in the product graph. There exists a transition $a \to c$ in the Markov chain, however there is no successor (c, t) in the SCC of the product graph, with t being a successor of q_a in

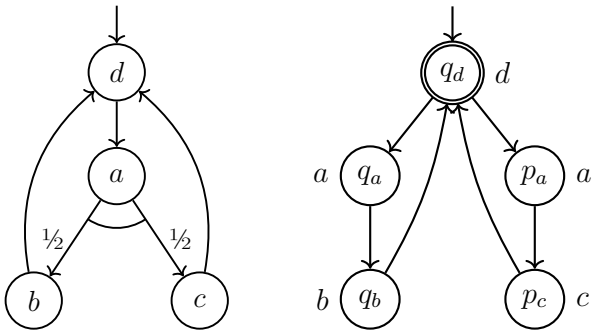


Figure 4.3: Markov chain $\mathcal{M}$ (left) and UBA $\mathcal{U}$ (right, state labels from alphabet $\{a, b, c, d\}$).

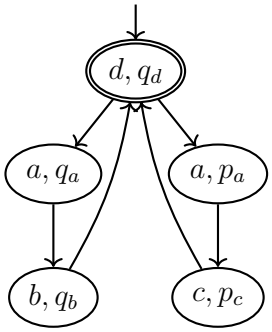


Figure 4.4: Product graph according to [BLW13b].

the UBA (c can not be consumed from the state q_a in the UBA). As the (only) SCC in the product is not accepting, all vertices in the product graph are assigned value 0 in the linear equation system, yielding that $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{U})) = 0$. However, as stated above, $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{U})) = 1$.

4.1.2 Counterexample for the approach of [BLW14]

Since [BLW13b] was flawed, the authors of [BLW13b] attempted to repair the proposed UBA-based analysis of Markov Chains [BLW14]. However, the approach of [BLW14] is flawed as well. We now present a detailed explanation for [BLW14].

They first present a technique for computing the probability of a Markov chain to satisfy a (co-)safety specification given by an unambiguous finite automaton (UFA) using a linear equation system with variables for pairs of states in the Markov chain and the UFA. This approach can be seen as an elegant variant of the universality test for UFA using difference equations [SH85].

In what follows, let $\mathcal{M} = (S, P, \iota)$ be a Markov chain and $\mathcal{U} = (Q, \Sigma, \delta, Q_0, \mathrm{Inf}(F))$ an UBA with the alphabet $\Sigma = S$. In this section, we use the following additional

notations:[2]

$$\Pr^{\#}_{\mathcal{M}[s]} \;=\; \sum_{t\in S} P(s,t)\cdot \Pr_{\mathcal{M}[t]}$$

The task addressed in [BLW14] is to compute $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{U}))$. The algorithm proposed in [BLW14] relies on the mistaken belief that if the Markov chain $\mathcal{M}$ generates words accepted by the given UBA $\mathcal{U}$ with positive probability then the product-graph $\mathcal{M}\otimes\mathcal{U}$ contains *recurrent pairs*. These are pairs $\langle s,q\rangle$ consisting of a state s in $\mathcal{M}$ and a state q of $\mathcal{U}$ such that almost all paths in $\mathcal{M}$ starting in a successor of s can be written as the infinite concatenation of cycles around s that have a run in $\mathcal{U}$ starting and ending in q. (The formal definition of recurrent pairs will be given below.) This claim, however, is wrong as there exist UBA that continuously need a look-ahead for the paths starting in a fixed state of the Markov chain.

Before presenting a counterexample illustrating this phenomenon and the faultiness of [BLW14], we recall some notations of [BLW14]. Given a state $s\in S$ of the Markov chain $\mathcal{M}$ and a state $q\in Q$ of the UBA $\mathcal{U}$, the regular languages $G_{s,q}, H_{s,q}\subseteq S^+$ are defined as follows:

$$\begin{aligned} G_{s,q} &= \big\{\, s_1\,s_2\ldots s_k\in S^+ \,:\, s_k=s \text{ and } p \xrightarrow{s_1\,s_2\ldots\,s_k} q \text{ for some } p\in Q_0 \,\big\}\\ H_{s,q} &= \big\{\, s_1\,s_2\ldots s_k\in S^+ \,:\, s_k=s \text{ and } q \xrightarrow{s_1\,s_2\ldots\,s_k} q \,\big\}\end{aligned}$$

A pair $\langle s,q\rangle\in S\times F$ is called *recurrent* if $\Pr^{\#}_{\mathcal{M}[s]}\big(H^\omega_{s,q}\big)=1$.

The accepted language of $\mathcal{U}$ can then be written as:

$$\mathcal{L}_\omega(\mathcal{U}) \;=\; \bigcup_{(s,q)\in S\times F} G_{s,q}\cdot H^\omega_{s,q}$$

The idea of [BLW14] is to reduce the task to compute $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{U}))$ to the task of computing the probability for $\mathcal{M}$ to generate a finite word accepted by a UFA for the language given by the UFA resulting from the union of the regular languages $G_{s,q}$ where $\langle s,q\rangle$ is recurrent. To show the correctness of this approach, [BLW14] claims that for each pair $\langle s,q\rangle\in S\times Q$:

$$\Pr^{\#}_{\mathcal{M}[s]}\big(\,H^\omega_{s,q}\,\big) \;\in\; \big\{0,1\big\}$$

and thus $\Pr^{\#}_{\mathcal{M}[s]}\big(H^\omega_{s,q}\big)=0$ for the non-recurrent pairs $\langle s,q\rangle\in S\times F$. To prove this claim, the authors conjecture (in Equation (5) of [BLW14]) that:

$$\Pr^{\#}_{\mathcal{M}[s]}\big(\,(H_{s,q})^\omega\,\big) \;=\; \lim_{n\to\infty}\Pr^{\#}_{\mathcal{M}[s]}\big(\,(H_{s,q})^n\,\big) \;\overset{(*)}{=}\; \lim_{n\to\infty}\Pr^{\#}_{\mathcal{M}[s]}\big(\,H_{s,q}\,\big)^n$$

The following example shows that equality $(*)$ is wrong, and recurrent pairs need not exist, even if $\mathcal{U}$ is universal.

[2] We depart here from the notations of [BLW14] where the notation $\Pr_{\mathcal{M},t}$ has been used as a short form for $\Pr^{\#}_{\mathcal{M}[t]}$.

Example. We consider the Markov chain $\mathcal{M} = (S, P, \iota)$ with two states, say $S = \{a, b\}$, and the transition probabilities

$$P(a,a) \;=\; P(a,b) \;=\; P(b,a) \;=\; P(b,b) \;=\; \tfrac{1}{2}$$

and the uniform initial distribution, i.e., $\iota(a) = \iota(b) = \frac{1}{2}$ (depicted in Figure 4.5 on the left). Thus:

$$\Pr^{\#}_{\mathcal{M}[a]} \;=\; \Pr^{\#}_{\mathcal{M}[b]} \;=\; \tfrac{1}{2} \cdot \Pr_{\mathcal{M}[a]} + \tfrac{1}{2} \cdot \Pr_{\mathcal{M}[b]}$$

From state a, the Markov chain $\mathcal{M}$ schedules almost surely an infinite word w starting with a and containing both symbols a and b infinitely often. The analogous statement holds for state b of $\mathcal{M}$.

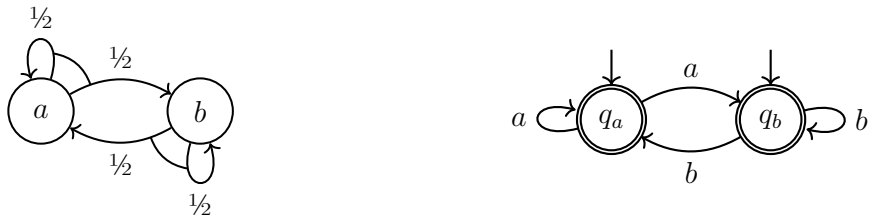


Figure 4.5: Markov chain $\mathcal{M}$ (left) and universal UBA $\mathcal{U}$ (right).

Let $\mathcal{U} = \big(Q, \{a,b\}, \delta, Q, \mathrm{Inf}(Q)\big)$ be the UBA with state space $Q = \{q_a, q_b\}$ where both states are initial and final and

$$\delta(q_a, a) \;=\; \delta(q_b, b) \;=\; \big\{q_a, q_b\big\},$$

while $\delta(q_a, b) = \delta(q_b, a) = \varnothing$ (depicted in Figure 4.5 on the right). Then, $\mathcal{U}$ is universal as $\mathcal{U}$ can use a one-letter look-ahead to generate an infinite run for each infinite word over $\{a, b\}$. More precisely, for doing so, $\mathcal{U}$ moves to state q_a if the next letter is a and to state q_b is the next letter is b. As both states are final, each word has an accepting run. Thus, $\mathcal{L}_\omega(\mathcal{U}) = \{a,b\}^\omega$ and therefore $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{U})) = 1$.

The language H_{a,q_a} is given by the regular expression $a((a + b^*)a)^*$. Note that the first a stems from the fact that we start in q_a in $\mathcal{U}$ and can only consume a. The a from the end of the regular expression stems from the definition of H_{a,q_a}, where the last symbol has to agree with the Markov chain state, in this case a. Thus, for $n \geqslant 2$, the language H^n_{a,q_a} consists of all finite words $x \in \{a,b\}^*$ that start with letter a and contain at least n occurrences of two consecutive letters a. Likewise, the language H^ω_{a,q_a} consists of all infinite words over $\{a,b\}$ with infinitely many aa's and where the first letter is a. Hence:

$$\begin{aligned} \Pr_{\mathcal{M}[a]}\big(H^\omega_{a,q_a}\big) \;&=\; \Pr_{\mathcal{M}[a]}\big(H^n_{a,q_a}\big) \;=\; 1 \\ \Pr_{\mathcal{M}[b]}\big(H^\omega_{a,q_a}\big) \;&=\; \Pr_{\mathcal{M}[b]}\big(H^n_{a,q_a}\big) \;=\; 0 \end{aligned}$$

for all $n \in \mathbb{N}$ with $n \geqslant 1$. This yields:

$$\Pr^{\#}_{\mathcal{M}[a]}\big(\, H^{\omega}_{a,q_a} \,\big) \;=\; \Pr^{\#}_{\mathcal{M}[a]}\big(\, H^{n}_{a,q_a} \,\big) \;=\; \tfrac{1}{2}$$

for all $n \in \mathbb{N}$ with $n \geqslant 1$. On the other hand:

$$\lim_{n\to\infty} \Pr^{\#}_{\mathcal{M}[a]}(\, H_{a,q_a} \,)^n \;=\; \lim_{n\to\infty} \big(\tfrac{1}{2}\big)^n \;=\; 0$$

Thus, equality $(*)$ is wrong. In this example, none of the pairs $\langle a, q_a\rangle$, $\langle a, q_b\rangle$, $\langle b, q_a\rangle$, $\langle b, q_b\rangle$ is recurrent. Note that the languages H_{a,q_b} and H_{b,q_a} are empty and that an analogous calculation yields:

$$\Pr^{\#}_{\mathcal{M}[b]}\big(\, H^{\omega}_{b,q_b} \,\big) \;=\; \Pr^{\#}_{\mathcal{M}[b]}\big(\, H^{n}_{b,q_b} \,\big) \;=\; \tfrac{1}{2}$$

and $\lim\limits_{n\to\infty} \Pr^{\#}_{\mathcal{M}[b]}(\, H_{b,q_b} \,)^n = 0$.

4.1.3 Strongly connected UBA

In Section 4.1.3 we prove Theorem 4.1 in the special case for a uniform Markov chain and a UBA, which is strongly connected. Therefore, we abbreviate $\Pr_{\mathcal{M}}(L)$ by $\Pr(L)$ if $\mathcal{M}$ is the uniform Markov chain and L an ω-regular language. We generalize the results in Section 4.1.4 and Section 4.1.5.

For this we start with a lemma about positive ω-regular languages which we use very often, in particular in case of strongly connected UBA.

Lemma 4.2. *If $L \subseteq \Sigma^\omega$ is ω-regular and $\Pr(L) > 0$, then there exists $x \in \Sigma^*$ such that $\Pr\big(\{\, w \in \Sigma^\omega \,:\, xw \in L \,\}\big) = 1$.*

Proof. Pick a deterministic ω-automaton $\mathcal{D} = (Q, \Sigma, \delta, q_{\mathit{init}}, \Phi)$ for L, for instance, with a Rabin acceptance condition. W.l.o.g. all states are reachable from q_{init} and $\mathcal{D}$ is complete. Let $\mathcal{M}_{\mathcal{D}} = (Q, P)$ be the transition-labeled Markov chain resulting from $\mathcal{D}$ by turning all branchings in $\mathcal{D}$ into uniform probabilistic choices, i.e., for each state q and each letter a, $P(q, a, q') = 1/|\Sigma|$ if $\delta(q, a) = q'$ and $P(q, a, q') = 0$ otherwise. Clearly, the underlying graph of $\mathcal{D}$ and $\mathcal{M}_D$ is the same. If $\mathcal{C}$ is a bottom strongly connected component (BSCC) of $\mathcal{D}$ resp. $\mathcal{M}_D$, then $\mathcal{C}$ is said to satisfy $\mathcal{D}$'s acceptance condition Φ, denoted $\mathcal{C} \models \Phi$, iff all infinite paths π with $\mathit{inf}(\pi) = \mathcal{C}$ meet the condition imposed by Φ, where $\mathit{inf}(\pi)$ denotes the set of states that appear infinitely often in π. For example, if Φ is a Rabin condition, say

$$\Phi \;=\; \bigvee_{1\leqslant i\leqslant k} \big(\, \mathrm{Fin}\,(U_i) \,\wedge\, \mathrm{Inf}\,(L_i) \,\big) \qquad \text{where} \qquad U_i, L_i \subseteq Q,$$

and $\mathcal{C} \subseteq Q$ a BSCC, then $\mathcal{C} \models \Phi$ iff there is at least one Rabin pair $\mathrm{Fin}\,(U_i) \wedge \mathrm{Inf}\,(L_i)$ in Φ with $\mathcal{C} \subseteq Q \setminus U_i$ and $\mathcal{C} \cap L_i \neq \varnothing$. As almost all paths in $\mathcal{M}_{\mathcal{D}}$ eventually enter a BSCC and visit all its states infinitely often, we get:

$$\begin{aligned} \Pr(L) > 0 \quad &\text{iff} \quad \Pr_{\mathcal{M}_{\mathcal{D}}}(\Phi) > 0 \\ &\text{iff} \quad \mathcal{D} \text{ has at least one BSCC } \mathcal{C} \text{ with } \mathcal{C} \models \Phi \end{aligned}$$

In this case and if x is a finite word such that $\delta(q_{init}, x) \cap \mathcal{C} \neq \varnothing$, then $\Pr(\{\, w \in \Sigma^\omega : xw \in L \,\}) = 1$.

□

We continue with some general observations about strongly connected Büchi automata under probabilistic semantics. For this, we suppose $\mathcal{A} = \big(Q, \Sigma, \delta, Q_0, \mathrm{Inf}(F)\big)$ is a strongly connected NBA where Q_0 and F are non-empty. Note that $\mathcal{A}$ might be incomplete.

Fact 4.3. *Suppose $\mathcal{A}$ is a strongly connected NBA. Then, the following statements are equivalent:*

(1) $\Pr(\mathcal{L}_\omega(\mathcal{A})) > 0$

(2) $\Pr(\mathcal{L}_\omega(q)) > 0$ *for some state q*

(3) $\Pr(\mathcal{L}_\omega(p)) > 0$ *for all states p*

Proof. The implications (1) $\Longrightarrow$ (2) and (3) $\Longrightarrow$ (1) are trivial. We now show that (2) $\Longrightarrow$ (3). Since $\mathcal{A}$ is strongly connected, there exists a finite word x with $p \xrightarrow{x} q$, i.e., $q \in \delta(p, x)$. But then

$$\Pr(\mathcal{L}_\omega(p)) \;\geqslant\; \frac{1}{|\Sigma|^{|x|}} \cdot \Pr(\mathcal{L}_\omega(q)) \;>\; 0$$

Note that $1/|\Sigma|^{|x|}$ is the probability for (the cylinder set spanned by) the finite word x.

□

Moreover, almost all words $w \in \Sigma^\omega \setminus \mathcal{L}_\omega(\mathcal{A})$ have a finite prefix x with $\delta(Q_0, x) = \varnothing$:

Lemma 4.4 (Measure of strongly connected NBA)**.** *For each strongly connected NBA $\mathcal{A}$ with at least one final state, we have:*

$$\Pr(\mathcal{L}_\omega(\mathcal{A})) \;=\; 1 - \Pr\big(\{\, w \in \Sigma^\omega : w \textit{ has a finite prefix } x \textit{ with } \delta(Q_0, x) = \varnothing \,\}\big)$$

In particular, $\mathcal{A}$ is almost universal if and only if $\delta(Q_0, x) \neq \varnothing$ for all finite words $x \in \Sigma^*$. This observation will be crucial at several places in the soundness proof of our algorithm for UBA, but can also be used to establish PSPACE-hardness of the positivity (probabilistic non-emptiness) and almost universality problem for strongly connected NBA (see Theorem 4.43).

To prove Lemma 4.4, we accumulate several statements, i.e., Lemma 4.5 until Lemma 4.11 serve as base for a proof. We start with showing that the language consisting of all words $w \in \Sigma^\omega \setminus \mathcal{L}_\omega(\mathcal{A})$ such that $\delta(Q_0, x) \neq \varnothing$ for all $x \in \mathrm{Pref}(w)$ is a null set. If $\mathcal{L}_\omega(\mathcal{A})$ has positive measure, this statement is a simple consequence of Lemma 4.2 (see Lemma 4.5 below). The general case will be shown in Lemma 4.11 using known results for the positive probabilistic model checking problem.

Lemma 4.5. *Suppose $\mathcal{A}$ is strongly connected and $\Pr(\mathcal{L}_\omega(\mathcal{A})) > 0$. Let L denote the set of infinite words $w \in \Sigma^\omega \setminus \mathcal{L}_\omega(\mathcal{A})$ such that $\delta(Q_0, x) \neq \varnothing$ for all $x \in \mathrm{Pref}(w)$. Then, $\Pr(L) = 0$.*

Proof. Suppose by contradiction that $\Pr(L)$ is positive. Obviously, L is ω-regular. Lemma 4.2 yields the existence of a finite word x such that

$$\Pr\big(\{\ v \in \Sigma^\omega\ :\ xv \in L\ \}\big) \quad = \quad 1$$

Let $R = \delta(Q_0, x)$. Then, R is non-empty and $\Pr(\,\mathcal{L}_\omega(\mathcal{A}[R])\,) = 0$, i.e., $\Pr(\,\mathcal{L}_\omega(q)\,) = 0$ for all states $q \in R$. This is impossible by Fact 4.3.

□

To prove the analogous result for the general case (possibly $\Pr(\mathcal{L}_\omega(\mathcal{A})) = 0$), we rely on results by Courcoubetis and Yannakakis [CY95] for the positive probabilistic model checking problem. These results rephrased for our purposes yield the following. Let $\mathcal{A}_{\text{det}}$ denote the standard powerset construction of $\mathcal{A}$. That is, the states of $\mathcal{A}_{\text{det}}$ are the subsets of Q and the transitions in $\mathcal{A}_{\text{det}}$ are given by $R \xrightarrow{a} R'$ iff $R' = \delta(R, a)$. The initial state of $\mathcal{A}_{\text{det}}$ is Q_0. $\mathcal{A}_{\text{det}}$ is viewed here just as a pointed labeled graph rather than an automaton over words.

Recall that $\mathcal{A}$ might be incomplete. Thus, $\varnothing$ is a trap state of $\mathcal{A}_{\text{det}}$ that is reached via the a-transition from any state $R \subseteq Q$ where $\delta(R, a)$ is empty. Hence, $\{\varnothing\}$ is a BSCC of $\mathcal{A}_{\text{det}}$ that might or might not be reachable from Q_0. We refer to $\{\varnothing\}$ as the *trap-BSCC* of $\mathcal{A}_{\text{det}}$. All other BSCCs of $\mathcal{A}_{\text{det}}$ are called *non-trap*.

A state $q \in Q$ of $\mathcal{A}$ is said to be *recurrent*[3] if there is some BSCC $\mathcal{C}$ of $\mathcal{A}_{\text{det}}$ that contains a state $R \subseteq Q$ of $\mathcal{A}_{\text{det}}$ with $q \in R$ and that is reachable from the singleton $\{q\}$ viewed as a state of $\mathcal{A}_{\text{det}}$. That is, q is recurrent iff there exists a finite word x such that $q \xrightarrow{x} q$ and the set $\delta(q, x)$ belongs to a BSCC of $\mathcal{A}_{\text{det}}$.

Fact 4.6 (Proposition 4.1.4 in [CY95])**.** *For each NBA $\mathcal{A}$ (not necessarily strongly connected):*

$$\Pr\big(\,\mathcal{L}_\omega(\mathcal{A})\,\big) > 0 \qquad \textit{iff} \qquad \begin{cases} \textit{there exists a finite word } x \in \Sigma^* \textit{ such that} \\ \delta(Q_0, x) \cap F \textit{ contains a recurrent state} \end{cases}$$

Example 4.7. *We consider the strongly connected NBA shown in Figure 4.6. Then, $\mathcal{A}_{det}$ has two BSCCs, namely the trap-BSCC $\{\varnothing\}$ and the non-trap BSCC $\{Q\}$. The singletons $\{q_a\}$ and $\{q_b\}$, viewed as states of $\mathcal{A}_{det}$, can reach $\{Q\}$. Hence, both states q_a and q_b are recurrent. To justify the statement of Fact 4.6, we observe that $\Pr(\mathcal{L}_\omega(\mathcal{A})) = 1/2 > 0$ and $\delta(\{q_a\}, a) \cap F = \{q_b\}$ contains a recurrent state.*

[3]The word recurrent has two meanings in this thesis. For the counterexamples of [BLW13b; BLW14] the term is used in the meaning of the according papers, whereas from now on we use the term recurrent in the sense of a state q of $\mathcal{A}$ such that a state R with $q \in R$ is reachable from $\{q\}$ in $\mathcal{A}_{\text{det}}$.

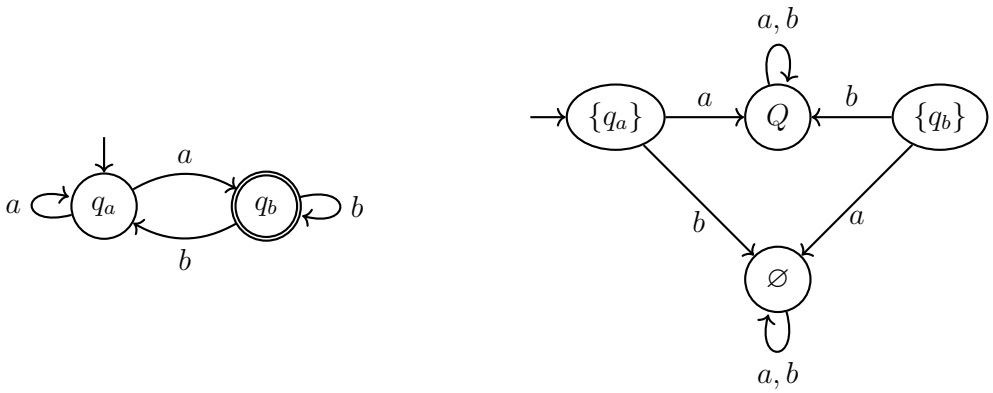


Figure 4.6: Automata from Example 4.7, NBA $\mathcal{A}$ (left) and corresponding $\mathcal{A}_{\text{det}}$ (right).

Suppose now that $\mathcal{A}$ is strongly connected. Then, for each non-trap BSCC $\mathcal{C}$ of $\mathcal{A}_{\text{det}}$ (i.e., $\mathcal{C} \neq \{\varnothing\}$) and each state p of $\mathcal{A}$ there exists some $R \subseteq Q$ with $p \in R \in \mathcal{C}$. Moreover, whenever $R \in \mathcal{C}$ and $x \in \Sigma^*$, then $\delta(R, x) \in \mathcal{C}$.

Lemma 4.8. *If $\mathcal{A}$ is strongly connected and $\mathcal{A}_{det}$ has a non-trap BSCC that is reachable from some singleton $\{q\}$, then all states $p \in Q$ are recurrent.*

Proof. Let $\mathcal{C}$ be a non-trap BSCC of $\mathcal{A}_{\text{det}}$ that is reachable from $\{q\}$ and let p be a state of $\mathcal{A}$. We pick finite words $x, y \in \Sigma^*$ such that $q \in \delta(p, x)$ and $\delta(q, y) \in \mathcal{C}$. Then, for all finite words z, $\delta(q, yz) \in \mathcal{C}$ and therefore:

$$\varnothing \quad \neq \quad \delta(q, yz) \quad \subseteq \quad \delta(p, xyz)$$

Hence, $\delta(p, xyz) = \delta(\delta(p, xy), z)$ is non-empty for all words z. Thus, there is a non-trap BSCC $\mathcal{C}'$ (possibly different from $\mathcal{C}$) that is reachable from p via some finite word of the form xyz. As stated above, $\mathcal{C}'$ contains some $R \subseteq Q$ with $p \in R$. Hence, p is recurrent. □

We summarize Fact 4.6 and Lemma 4.8 in the following corollary:

Corollary 4.9 (Probabilistic emptiness of strongly connected NBA)**.** *Let $\mathcal{A}$ be a strongly connected NBA with at least one final state. Then, the following statements are equivalent:*

(1) $\Pr\big(\mathcal{L}_\omega(\mathcal{A})\big) = 0$.

(2) *$\mathcal{A}_{det}$ has no non-trap BSCC that is reachable from some singleton $\{q\}$.*

(3) *There is no recurrent state in $\mathcal{A}$.*

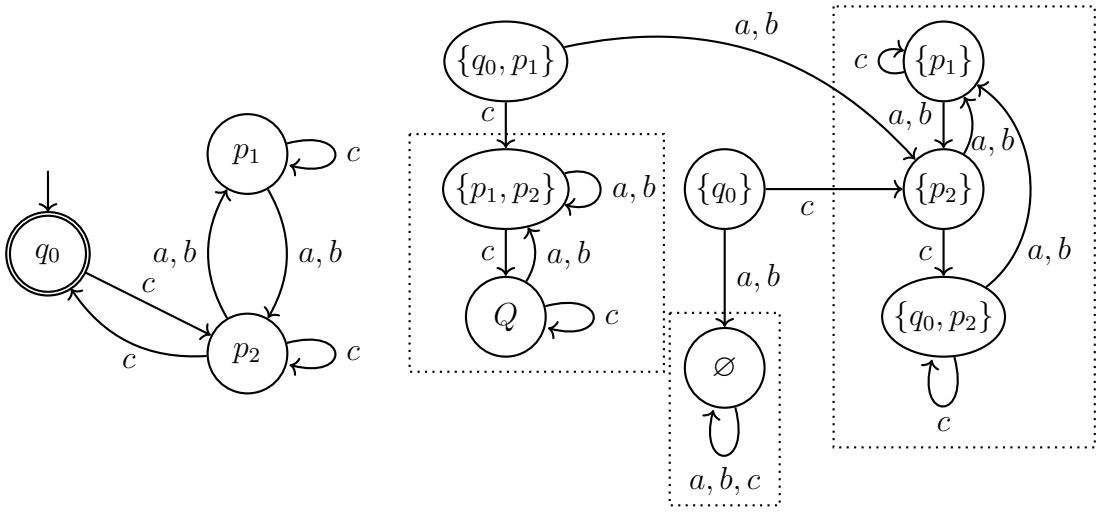


Figure 4.7: Automata from Example 4.10, NBA $\mathcal{A}$ (left) and $\mathcal{A}_{\text{det}}$ (right, with BSCCs).

Example 4.10. *We consider the strongly connected NBA shown in Figure 4.7. Then, $\mathcal{A}_{det}$ has a non-trap BSCC consisting of the states $\{p_1, p_2\}$ and Q that is not accessible from any singleton. However, there is another non-trap BSCC in $\mathcal{A}_{det}$ consisting of the three states $\{p_1\}$, $\{p_2\}$ and $\{q_0, p_2\}$. Indeed, $\mathcal{A}$ accepts almost all words starting with letter c and therefore* $\Pr(\mathcal{L}_\omega(\mathcal{A})) = 1/3$.

We are now ready to complete the proof of Lemma 4.4 by proving the following lemma:

Lemma 4.11. *Suppose $\mathcal{A}$ is strongly connected with at least one final state. Let L be the set of infinite words $w \in \Sigma^\omega \setminus \mathcal{L}_\omega(\mathcal{A})$ such that $\delta(Q_0, x) \neq \varnothing$ for all $x \in \text{Pref}(w)$. Then,* $\Pr(L) = 0$.

Proof. We consider first the case where $\mathcal{A}$ has a single initial state, say $Q_0 = \{q_0\}$. Suppose by contradiction that $\Pr(L)$ is positive. Then, there is some finite word z such that $zv \in L$ for almost all words $v \in \Sigma^\omega$. Let $R = \delta(q_0, z)$. By definition of L, the set R is non-empty and $\delta(R, x) \neq \varnothing$ for all finite words x. Hence, the state $\varnothing$ is not reachable from R in $\mathcal{A}_{\text{det}}$. Therefore, there is a non-trap BSCC of $\mathcal{A}_{\text{det}}$ that is reachable from the singleton $\{q_0\}$. Hence, $\Pr(\mathcal{L}_\omega(\mathcal{A})) > 0$ by Corollary 4.9. But then $\Pr(L) = 0$ by Lemma 4.5. Contradiction.

The argument for the general case is as follows. Suppose by contradiction that L has positive measure. We consider the labeled Markov chain $\mathcal{M} = (2^Q, P, \mathsf{Dirac}[Q_0])$ that arises from the deterministic automaton $\mathcal{A}_{\text{det}}$ with initial state Q_0 by attaching uniform distributions. That is, if $R, R' \subseteq Q$ and $a \in \Sigma$, then $P(R, a, R') = 1/|\Sigma|$ if $R' = \delta(R, a)$ and $P(R, a, R') = 0$ otherwise. For almost all words w in L, the corresponding path π_w in $\mathcal{M}$ eventually visits some BSCC $\mathcal{C}$ of $\mathcal{M}$ resp. $\mathcal{A}_{\text{det}}$ and visits all its states infinitely often. By assumption $\mathcal{C}$ is non-trap.

The goal is to show that a non-trap BSCC is accessible from some singleton. Let L' denote the set of all words $w = a_1\, a_2\, a_3 \ldots \in L$ such that π_w eventually enters some BSCC of $\mathcal{M}$ and visits all its states infinitely often. Then, $\Pr(L') = \Pr(L)$. If $w = a_1\, a_2\, a_3 \ldots \in L'$ and $\pi_w = R_0\, R_1\, R_2 \ldots$ then $R_0 = Q_0$ and $R_n = \delta(Q_0, a_1 \ldots a_n)$. By König's Lemma (see, e.g., Lemma 8.1.2 of [Die17]) there is an infinite run $q_0\, q_1\, q_2 \ldots$ for w in $\mathcal{A}$ such that $q_i \in R_i$ for all $i \in \mathbb{N}$. We write $\mathit{Runs}(w)$ to denote the set of all these runs. Pick some run $\rho = q_0\, q_1\, q_2 \ldots \in \mathit{Runs}(w)$ and define $U_0 = \{q_0\}$ and $U_i = \delta(q_0, a_1\, a_2 \ldots a_i) = \delta(U_{i-1}, a_i)$ for $i \geqslant 1$. Clearly, $q_i \in U_i \subseteq R_i$ for all indices i and $\pi_\rho = U_0\, U_1\, U_2 \ldots$ is a path in $\mathcal{M}$ and the unique run for w in $\mathcal{A}_{\mathrm{det}}$ starting in $\{q_0\}$.

Consider the set $\mathfrak{U}$ of all sets $U \subseteq Q$ such that $U \in \mathit{inf}(\pi_\rho)$ for some word $w \in L'$ and some $\rho \in \mathit{Runs}(w)$. (The notation $\mathit{inf}(\pi)$ is used to denote the set of elements that appear infinitely often in π.) We now show that the subgraph of $\mathcal{A}_{\mathrm{det}}$ consisting of the nodes $U \in \mathfrak{U}$ contains a BSCC. For each subset $\mathcal{V}$ of $\mathfrak{U}$ and each state $q \in Q$, let

$$L_{q,\mathcal{V}} \quad = \quad \left\{\, w \in L' \,:\, \exists \rho \in \mathit{Runs}(w) \text{ s.t. } q = \mathit{first}(\pi_\rho) \wedge \mathit{inf}(\pi_\rho) = \mathcal{V} \,\right\}$$

Then, L' is the union of all sets $L_{q,\mathcal{V}}$ with $(q, \mathcal{V}) \in Q_0 \times \mathfrak{U}$. As $\Pr(L') = \Pr(L) > 0$ and $Q_0 \times \mathfrak{U}$ is finite, there is some pair $(q, \mathcal{V}) \subseteq Q_0 \times \mathfrak{U}$ with $\Pr(L_{q,\mathcal{V}}) > 0$. Clearly, $L_{q,\mathcal{V}}$ is ω-regular. Hence, there is some finite word z such that

$$\Pr\left(\left\{\, v \in \Sigma^\omega \,:\, zv \in L_{q,\mathcal{V}} \,\right\}\right) \quad = \quad 1.$$

Let $R = \delta(q, z)$. We now regard the fragment of $\mathcal{A}_{\mathrm{det}}$ that is reachable from R. Let $\mathcal{M}[R]$ be the corresponding Markov chain (i.e., the sub Markov chain of $\mathcal{M}$ with initial state R). Since $zv \in L_{q,\mathcal{V}}$ for almost all words $v \in \Sigma^\omega$, $\mathit{inf}(\pi) = \mathcal{V}$ for almost all paths in $\mathcal{M}[R]$. But then $\mathcal{V}$ constitutes a non-trap BSCC of $\mathcal{M}[R]$, and therefore of $\mathcal{M}$ and $\mathcal{A}_{\mathrm{det}}$. Since $\mathcal{V}$ is reachable from $\{q\}$ in $\mathcal{A}_{\mathrm{det}}$, we obtain $\Pr(\mathcal{L}_\omega(\mathcal{A})) > 0$ by Corollary 4.9. Lemma 4.5 yields $\Pr(L) = 0$, which contradicts the assumption $\Pr(L) > 0$.

□

Remark 4.12. *Clearly,* $\Pr(\mathcal{L}_\omega(\mathcal{A}))$ *depends on* Q_0 *as there might be state-letter pairs* (q, a) *where* $\delta(q, a)$ *is empty. However, Lemma 4.4 implies that if* $\mathcal{A}$ *is strongly connected with at least one final state, then* $\Pr(\mathcal{L}_\omega(\mathcal{A}))$ *does not depend on* F*.*

For computing $\Pr(\mathcal{L}_\omega(\mathcal{U}))$ given a UBA $\mathcal{U}$, it suffices to compute the values $\Pr(\mathcal{L}_\omega(q))$ for the (initial) states of $\mathcal{U}$ as we have

$$\Pr(\mathcal{L}_\omega(\mathcal{U})) \quad = \quad \sum_{q \in Q_0} \Pr(\mathcal{L}_\omega(q))$$

The following simple fact will be used at several places:

Fact 4.13. *Let $\mathcal{U}$ be a UBA with $\mathcal{L}_\omega(q) \neq \varnothing$ for each state $q \in Q$. Then, for each state $p \in Q$, each non-empty subset R of Q of the form $R = \delta(Q_0, y)$ for some word $y \in \Sigma^*$ and each finite word $x = a_1\, a_2 \ldots a_n \in \Sigma^*$ there exists at most one state $q \in R$ and at most one run*

$$q = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ \ldots \ \xrightarrow{a_n} q_n = p$$

for x starting in q and ending in p. In particular, the NFA $(Q, \Sigma, \delta, \delta(Q_0, y), \mathrm{Reach}\,(\{p\}))$ is unambiguous for each $y \in \Sigma^*$ and each state $p \in Q$.

Proof. If there would be two runs $q_0 \xrightarrow{y} q \xrightarrow{x} p$ and $q_0' \xrightarrow{y} q' \xrightarrow{x} p$ where $q, q' \in R$ and $q_0, q_0' \in Q_0$, then each word yxw with $w \in \mathcal{L}_\omega(p)$ would have two accepting runs. This is impossible by the unambiguity of $\mathcal{U}$ and the assumption that $\mathcal{L}_\omega(p)$ is non-empty for all states $p \in Q$.

□

Fact 4.13 will often be used in form of the statement that, for all states $q, p \in Q$ and all finite words x, there is at most one run for x from q to p.

We now suppose that $\mathcal{U}$ is a strongly connected UBA. Note that $\mathcal{L}_\omega(p) \neq \varnothing$ and $\Pr(\mathcal{L}_\omega(p)) = 0$ is possible. However, in this case $\Pr(\mathcal{L}_\omega(q)) = 0$ for all states q. The following theorem states that for strongly connected UBA $\mathcal{U}$, the accepting runs of almost all words in $\mathcal{L}_\omega(\mathcal{U})$ visit each state of $\mathcal{U}$ infinitely often. Although irrelevant for the soundness of our algorithm, we find that it reveals an interesting structural property of strongly connected UBA.

We use the following notations. For $w \in \mathcal{L}_\omega(\mathcal{U})$, we write $\mathit{accrun}(w)$ to denote the unique accepting run for w in $\mathcal{U}$. For $q_0\, q_1\, q_2 \ldots \in Q^\omega$, the set $\mathit{inf}(q_0\, q_1\, q_2 \ldots)$ denotes the set of all states $q \in Q$ such that $q = q_i$ for infinitely many indices i. Thus, if $w \in \mathcal{L}_\omega(\mathcal{U})$, then $\mathit{inf}(\mathit{accrun}(w))$ collects all states $q \in Q$ that appear infinitely often in the accepting run for w.

Theorem 4.14 (Measure of strongly connected UBA)**.** *If $\mathcal{U}$ is a strongly connected UBA with at least one final state, then:*

$$\Pr\big(\{\, w \in \mathcal{L}_\omega(\mathcal{U}) \,:\, \mathit{inf}(\mathit{accrun}(w)) = Q \,\}\big) \quad = \quad \Pr\big(\, \mathcal{L}_\omega(\mathcal{U}) \,\big)$$

Proof. To prove Theorem 4.14, we can rely on the following facts that hold for all events (measurable sets) L_i, L in each probability space:

- $\Pr(L_1) = \Pr(L_2) = 1$ iff $\Pr(L_1 \cap L_2) = 1$

  Hence, if $L_1, \ldots, L_n \subseteq L$ and $\Pr(L_1) = \ldots = \Pr(L_n) = \Pr(L)$, then $\Pr(L_1 \cap \ldots \cap L_n) = \Pr(L)$ as $\Pr(L_i \mid L) = 1$ for all $i = 1, \ldots, n$ implies $\Pr(L_1 \cap \ldots \cap L_n \mid L) = 1$.

- if $(L_n)_{n \in \mathbb{N}}$ is a countable family of measurable sets with $L_0 \supseteq L_1 \supseteq L_2 \supseteq \ldots$, then

  $$\Pr(L) = \lim_{n \to \infty} \Pr(L_n) \qquad \text{where} \qquad L = \bigcap_{n \in \mathbb{N}} L_n$$

  as every probability measure is continuous from above.

Hence, for the proof of Theorem 4.14 it suffices to show that the accepting runs for almost all words in $\mathcal{L}_\omega(\mathcal{U})$ contain a fixed state q that appears infinitely often. Thus, the goal is to show:

$$\Pr\big(\{\, w \in \mathcal{L}_\omega(\mathcal{U}) \,:\, q \in \mathit{inf}(\mathit{accrun}(w)) \,\}\big) \;=\; \Pr\big(\,\mathcal{L}_\omega(\mathcal{U})\,\big)$$

The claim is obvious if $\Pr(\mathcal{L}_\omega(\mathcal{U})) = 0$. Suppose now that $\Pr(\mathcal{L}_\omega(\mathcal{U})) > 0$. Hence, $\Pr(\mathcal{L}_\omega(p)) > 0$ for all states p (Fact 4.3). We first show that the accepting runs for almost all words in $\mathcal{L}_\omega(\mathcal{U})$ eventually visit q.

Claim: $\Pr(\{\, w \in \mathcal{L}_\omega(\mathcal{U}) \,:\, \mathit{accrun}(w) \models \Diamond q \,\}) \;=\; \Pr\big(\,\mathcal{L}_\omega(\mathcal{U})\,\big)$.

Proof of the claim: Suppose by contradiction that there is some state $q \in Q$ such that the set of words $w \in \mathcal{L}_\omega(\mathcal{U})$ whose run does not visit q has positive measure, i.e.,

$$\Pr\{\, w \in \mathcal{L}_\omega(\mathcal{U}) \,:\, \mathit{accrun}(w) \not\models \Diamond q \,\} \;>\; 0$$

Lemma 4.2 yields the existence of a finite word $x \in \Sigma^*$ such that:

$$\Pr\big(\{\, w \in \Sigma^\omega \,:\, xw \in \mathcal{L}_\omega(\mathcal{U}),\ \mathit{accrun}(xw) \not\models \Diamond q \,\}\big) \;=\; 1$$

In particular, $\delta(Q_0, x)$ is non-empty. Pick some state $p \in \delta(Q_0, x)$, say $p \in \delta(q_0, x)$ where $q_0 \in Q_0$. As $\mathcal{U}$ is strongly connected, there is some non-empty finite word y with $q \in \delta(p, y)$. Let

$$L \;=\; \{\, xyv \,:\, v \in \mathcal{L}_\omega(q) \,\} \quad\text{and}\quad L' \;=\; \{\, yv \,:\, v \in \mathcal{L}_\omega(q) \,\}.$$

As $\Pr(\mathcal{L}_\omega(q)) > 0$ and there is exactly one run for xy from Q_0 to q, we get:

$$\Pr(L') \;\geqslant\; \Pr(L) \;=\; \frac{1}{|\Sigma|^n} \cdot \Pr(\mathcal{L}_\omega(q)) \;>\; 0$$

where $n = |x| + |y|$. The words $xyv \in L$ have accepting runs starting with the prefix:

$$q_0 \xrightarrow{\;x\;} p \xrightarrow{\;y\;} q$$

In particular:

$$L' \;\subseteq\; \{\, w \in \Sigma^\omega \,:\, xw \in \mathcal{L}_\omega(\mathcal{U}),\ \mathit{accrun}(xw) \models \Diamond q \,\}$$

and therefore:

$$\Pr\big(\{\, w \in \Sigma^\omega \,:\, xw \in \mathcal{L}_\omega(\mathcal{U}),\ \mathit{accrun}(xw) \models \Diamond q \,\}\big) \;>\; 0$$

But then:

$$\Pr\big(\{\, w \in \Sigma^\omega \,:\, xw \in \mathcal{L}_\omega(\mathcal{U}),\ \mathit{accrun}(xw) \not\models \Diamond q \,\}\big) \;<\; 1$$

Contradiction. This completes the proof of the claim.

With an analogous argument we get that for each state $q \in Q$, $\Pr(\mathcal{L}_\omega(\mathcal{U})) = \Pr(L_n)$ where L_n is the set of all infinite words $w \in \mathcal{L}_\omega(\mathcal{U})$ such that $accrun(w)$ visits state q at least n times. It holds, that

$$\mathcal{L}_\omega(\mathcal{U}) \;=\; L_0 \;\supseteq\; L_1 \;\supseteq\; L_2 \;\supseteq\; \ldots \;\supseteq\; \bigcap_{n\in\mathbb{N}} L_n \;\stackrel{\text{def}}{=}\; L$$

Then, L is the set of all words $w \in \mathcal{L}_\omega(\mathcal{U})$ such that $q \in inf(accrun(w))$. Additionally, we have:

$$\Pr(L) \;=\; \lim_{n\to\infty} \Pr(L_n) \;=\; \Pr(\mathcal{L}_\omega(\mathcal{U}))$$

This completes the proof of Theorem 4.14.

□

Remark 4.15. *The only place where the proof of Theorem 4.14 uses the unambiguity of $\mathcal{U}$ is in the statement that L agrees with the set of infinite words w such that $q \in inf(accrun(w))$.*

Remark 4.16. *With analogous arguments one can show that the accepting runs of almost all words in $\mathcal{L}_\omega(\mathcal{U})$ contain each finite path of $\mathcal{U}$ infinitely often.*

Deciding positivity for strongly connected UBA.

The following lemma provides a criterion to check positivity of a strongly connected UBA in polynomial time using standard linear algebra techniques.

Lemma 4.17. *Let $\mathcal{U}$ be a strongly connected UBA with at least one initial and one final state, and*

$$(*) \qquad \zeta_q \;=\; \frac{1}{|\Sigma|} \cdot \sum_{\sigma\in\Sigma} \sum_{p\in\delta(q,\sigma)} \zeta_p \qquad \text{for all } q \in Q$$

Then, the following statements are equivalent:

(1) $\Pr(\mathcal{L}_\omega(\mathcal{U})) > 0$,

(2) the linear equation system (*) has a strictly positive solution, i.e., a solution $(\zeta_q^*)_{q\in Q}$ with $\zeta_q^* > 0$ for all $q \in Q$,

(3) the linear equation system (*) has a non-zero solution.

Given the strongly connected UBA $\mathcal{U}$ with at least one final state, we define a matrix $M \in [0,1]^{Q\times Q}$ by $M_{p,q} = \frac{1}{|\Sigma|} \cdot |\{a \in \Sigma : q \in \delta(p,a)\}|$ for all $p, q \in Q$. Since $\mathcal{U}$ is strongly connected, M is irreducible. We write $\rho(M)$ for the spectral radius of M. We will use the following lemma in the proof of Lemma 4.17.

Lemma 4.18. *We have $\rho(M) \leq 1$. Moreover, $\rho(M) = 1$ if and only if $\Pr(\mathcal{L}_\omega(\mathcal{U})) > 0$.*

Proof. For $p, q \in Q$ and $n \in \mathbb{N}$, let $E_{p,n,q} \subseteq \Sigma^\omega$ denote the event of all words $w = \sigma_1\sigma_2\ldots$ such that $q \in \delta(p, \sigma_1\sigma_2\ldots\sigma_n)$. Its probability under the uniform distribution on Σ^ω is an entry in the n-th power of M:

$$\Pr(E_{p,n,q}) = (M^n)_{p,q} \tag{4.1}$$

In particular, $M^n_{p,q} \leq 1$ for all n. From the boundedness of M^n it follows (e.g., by [HJ13, Corollary 8.1.33]) that $\rho(M) \leq 1$. The same result implies that

$$\begin{aligned} \rho(M) = 1 \quad &\iff \quad \limsup_{n\to\infty} (M^n)_{p,q} > 0 \quad \text{for all } p, q \in Q \\ &\iff \quad \limsup_{n\to\infty} (M^n)_{p,q} > 0 \quad \text{for some } p, q \in Q \end{aligned} \tag{4.2}$$

For the rest of the proof, fix some state $p \in Q$. By the observations from the beginning of Section 4.1.3 it suffices to show that $\Pr(\mathcal{L}_\omega(p)) > 0$ if and only if $\rho(M) = 1$. To this end, consider the event $E_{p,n} := \bigcup_{q\in Q} E_{p,n,q}$. Notice that $(E_{p,n})_{n\in\mathbb{N}}$ forms a decreasing family of sets. We have:

$$\begin{aligned} \Pr(\mathcal{L}_\omega(p)) &= \lim_{n\to\infty} \Pr(E_{p,n}) && \text{by Lemma 4.4} \\ &= \lim_{n\to\infty} \Pr\left(\bigcup_{q\in Q} E_{p,n,q}\right) && \text{by definition of } E_{p,n} \end{aligned} \tag{4.3}$$

Assuming that $\rho(M) = 1$, we show that $\Pr(\mathcal{L}_\omega(p)) > 0$. Let $q \in Q$. We have:

$$\begin{aligned} \Pr(\mathcal{L}_\omega(p)) &\geq \limsup_{n\to\infty} \Pr(E_{p,n,q}) && \text{by (4.3)} \\ &= \limsup_{n\to\infty} (M^n)_{p,q} && \text{by (4.1)} \\ &> 0 && \text{by (4.2)} \end{aligned}$$

Conversely, assuming that $\rho(M) < 1$, we show that $\Pr(\mathcal{L}_\omega(p)) = 0$.

$$\begin{aligned} \Pr(\mathcal{L}_\omega(p)) &= \lim_{n\to\infty} \Pr\left(\bigcup_{q\in Q} E_{p,n,q}\right) && \text{by (4.3)} \\ &\leq \limsup_{n\to\infty} \sum_{q\in Q} \Pr(E_{p,n,q}) && \text{union bound} \\ &= \limsup_{n\to\infty} \sum_{q\in Q} (M^n)_{p,q} && \text{by (4.1)} \\ &= 0 && \text{by (4.2)} \end{aligned}$$

This concludes the proof. □

Proof of Lemma 4.17. "(1) $\implies$ (2)": Suppose $\Pr(\mathcal{L}_\omega(\mathcal{U})) > 0$. Define the vector $(\zeta^*_q)_{q\in Q}$ with $\zeta^*_q = \Pr(\mathcal{L}_\omega(q))$. It holds that

$$\mathcal{L}_\omega(q) = \bigcup_{\sigma\in\Sigma} \bigcup_{p\in\delta(q,\sigma)} \{\sigma w : w \in \mathcal{L}_\omega(p)\}$$

Since $\mathcal{U}$ is unambiguous, the sets $\{\sigma w : w \in \mathcal{L}_\omega(p)\}$ are pairwise disjoint. So, the vector $(\zeta_q^*)_{q\in Q}$ is a solution to the equation system.

As $\Pr(\mathcal{L}_\omega(\mathcal{U})) > 0$ and $\mathcal{U}$ is strongly connected, the observation at the beginning of Section 4.1.3 yields that $\Pr(\mathcal{L}_\omega(q)) > 0$ for all states q. Thus, the vector $(\zeta_q^*)_{q\in Q}$ is strictly positive.

"(2) $\Longrightarrow$ (3)" holds trivially.

"(3) $\Longrightarrow$ (1)": Suppose ζ^* is a non-zero solution of the linear equation system. Then, $M\zeta^* = \zeta^*$. Thus, 1 is an eigenvalue of M. This yields $\rho(M) \geqslant 1$. But then $\rho(M) = 1$ and $\Pr(\mathcal{L}_\omega(\mathcal{U})) > 0$ by Lemma 4.18. □

Positivity check via rank computation. The first positivity check we present relies simply on the rank of $M - I$ with M being the matrix from Lemma 4.17. As the second positivity check delivers also a possible way to calculate $(\Pr(\mathcal{L}_\omega(q)))_{q\in Q}$ we postpone its presentation to the end of this section.

Let $\mathcal{U}$ be a strongly connected UBA with at least one final state and let M be the $n \times n$-matrix from Lemma 4.17, where $n = |Q|$ is the number of states in $\mathcal{U}$. We compute $\operatorname{rank}(M')$ of the matrix $M' = M - I$, where I is the identity matrix. If M' has full rank, i.e., $\operatorname{rank}(M') = n$, then $\mathcal{U}$ is non-positive.

If M' does not have full rank, we know that $\mathcal{U}$ is positive by the following argument. As M' does not have full rank n, there is a vector v such that $(M - I)v$ is the zero vector; in other words, v is an eigenvector of M with eigenvalue 1. So the spectral radius of M is at least 1. But by Lemma 4.18 the spectral radius of M is at most 1, so it is equal to 1. Since M is irreducible, it follows from the Perron-Frobenius theorem [BP79, Theorem 2.1.4 (b)] that M has a strictly positive eigenvector v' with eigenvalue 1, i.e., $Mv' = v'$. Lemma 4.17 then yields that $\mathcal{U}$ is positive.

Computing pure cuts for positive, strongly connected UBA.

The key observation to compute the values $\Pr(\mathcal{L}_\omega(q))$ for the states q of a positive, strongly connected UBA $\mathcal{U}$ is the existence of so-called cuts. These are sets C of states with pairwise disjoint languages such that almost all words have an accepting run starting in some state $q \in C$. More precisely:

Definition 4.19 ((Pure) cut)**.** *Let $\mathcal{U}$ be a UBA and $C \subseteq Q$. C is called a* cut *for $\mathcal{U}$ if $\mathcal{L}_\omega(q) \cap \mathcal{L}_\omega(p) = \varnothing$ for all $p, q \in C$ with $p \neq q$ and $\mathcal{U}[C]$ is almost universal. A cut is called* pure *if it has the form $\delta(q, z)$ for some state q and some finite word $z \in \Sigma^*$.*

Obviously, $\mathcal{U}$ is almost universal iff Q_0 is a cut. If $q \in Q$ and K_q denotes the set of finite words $z \in \Sigma^*$ such that $\delta(q, z)$ is a cut, then $\Pr(\mathcal{L}_\omega(q))$ equals the probability measure of the language L_q consisting of all infinite words $w \in \Sigma^\omega$ that have a prefix in K_q.

The following concept of cut languages is irrelevant for the soundness proof of our algorithm to compute $\Pr(\mathcal{L}_\omega(\mathcal{U}))$. However, we find that Lemma 4.21 (see below) is an interesting observation.

Definition 4.20 (Cut languages)**.** *For each state q in $\mathcal{U}$, let K_q be the set of finite words x such that $\delta(q,x)$ is a cut. We refer to K_q as the cut language for state q.*

The cut languages K_q are upward-closed (i.e., if $x \in K_q$, then $xy \in K_q$ for all finite words y) by the first statement of Fact 4.22. The second statement of Fact 4.22 yields that K_q is non-empty if $\Pr(\mathcal{L}_\omega(q))$ is positive. Vice versa, if $x \in K_q$, then almost all infinite words w with $x \in \mathrm{Pref}(w)$ belong to $\mathcal{L}_\omega(q)$. Hence, the cut language K_q is non-empty if and only if $\Pr(\mathcal{L}_\omega(q)) > 0$. Moreover, the cut language K_q is regular since $K_q = \mathcal{L}_{\mathrm{fin}}(\mathcal{U}_{\mathrm{det}}[q])$ where $\mathcal{U}_{\mathrm{det}}[q]$ denotes the DFA that results from the powerset construction of $\mathcal{U}$ by declaring state $\{q\}$ to be initial and the states that cannot reach the trap BSCC to be final (in particular, the states in the non-trap BSCCs are final). The unambiguity of $\mathcal{U}$ yields that $K_q \cap K_p = \varnothing$ for all states q, $p \in Q$ such that $\{q,p\} \subseteq \delta(Q_0,x)$ for some finite word $x \in \Sigma^*$.

We use $\Pr(K_q)$ as a short form notation for $\Pr(L)$ where L consists of all infinite words that have some prefix in K_q. Recall that the cut language K_q is empty if $\Pr(\mathcal{L}_\omega(q)) = 0$, in which case $\Pr(\mathcal{L}_\omega(q)) = \Pr(K_q) = 0$.

Lemma 4.21. $\Pr(\mathcal{L}_\omega(q)) \;=\; \Pr(K_q)$.

Proof. Obviously, almost all infinite words that have a prefix in K_q belong to $\mathcal{L}_\omega(q)$. This yields $\Pr(\mathcal{L}_\omega(q)) \geqslant \Pr(K_q)$. Suppose by contradiction that $\Pr(\mathcal{L}_\omega(q)) > \Pr(K_q)$. Then, the regular language $\{w \in \mathcal{L}_\omega(q) : \mathrm{Pref}(w) \cap K_q = \varnothing\}$ is positive. Lemma 4.2 yields the existence of some finite word x such that:

$$\Pr\Big(\big\{\, v \in \Sigma^\omega \;:\; xv \in \mathcal{L}_\omega(q),\ \mathrm{Pref}(xv) \cap K_q = \varnothing \,\big\}\Big) \;=\; 1$$

In particular, $x \notin K_q$ and $\Pr\big\{\, v \in \Sigma^\omega \;:\; xv \in \mathcal{L}_\omega(q) \,\big\} = 1$. The latter yields that $\delta(q,x)$ is a cut. But then $x \in K_q$ (by definition of K_q). Contradiction. □

The following facts are simple, general observations about cuts that will be used at various places:

Fact 4.22. *Suppose $\mathcal{U}$ is a UBA (possibly not strongly connected).*

- *If C is a cut, then so are the sets $\delta(C,y)$ for all finite words $y \in \Sigma^*$.*
- *If $\Pr(\mathcal{L}_\omega(\mathcal{U})) > 0$, then there exists some finite word $x \in \Sigma^*$ such that $\delta(Q_0,xy)$ is a cut for all words $y \in \Sigma^*$.*

Proof. The first statement is obvious. The argument for the second statement is as follows. Suppose $\Pr(\mathcal{L}_\omega(\mathcal{U})) > 0$. Lemma 4.2 asserts the existence of a finite word x such that

$$\Pr\big\{\, w \in \Sigma^\omega \;:\; xw \in \mathcal{L}_\omega(\mathcal{U}) \,\big\} \;=\; 1$$

Hence, the $\delta(Q_0,x)$ is a cut, and so are the sets $\delta(Q_0,xy) = \delta(\delta(Q_0,x),y)$ for all $y \in \Sigma^*$ by the first statement. □

Fact 4.22 yields the following characterization of almost universal UBA:

$$\begin{array}{lll} \mathcal{U} \text{ is almost universal} & \text{iff} & Q_0 \text{ is a cut} \\ & \text{iff} & \delta(Q_0, x) \text{ is a cut for all } x \in \Sigma^* \end{array}$$

The above characterization holds in any (possibly not strongly connected) UBA. An analogous characterization of positivity for strongly connected UBA is obtained using the fact that $\Pr(\mathcal{L}_\omega(\mathcal{U})) > 0$ iff $\Pr(\mathcal{L}_\omega(q)) > 0$ for some state q:

$$\begin{array}{ll} & \mathcal{U} \text{ is positive, i.e., } \Pr(\mathcal{L}_\omega(\mathcal{U})) > 0 \\ \text{iff} & \mathcal{U} \text{ has a reachable cut, i.e., a cut of the form } \delta(Q_0, x) \\ \text{iff} & \mathcal{U} \text{ has a pure cut, i.e., a cut of the form } \delta(q, x) \end{array}$$

We now elaborate the notion of cuts in strongly connected UBA.

Fact 4.23. *Suppose $\mathcal{U}$ is a strongly connected UBA and $\Pr(\,\mathcal{L}_\omega(\mathcal{U})\,) > 0$.*

- *Let C be a cut and C' a subset of Q with $\mathcal{L}_\omega(q) \cap \mathcal{L}_\omega(p) = \varnothing$ for all states $q, p \in C'$ with $q \neq p$. Then, $C \subseteq C'$ implies $C = C'$.*
- *If $\delta(q, x)$ is a cut and $y \in \Sigma^*$ such that $q \in \delta(q, xy)$, then $\delta(q, x) = \delta(q, xyx)$.*

Proof. The first statement is obvious as $\Pr(\mathcal{L}_\omega(r)) > 0$ for all states r. For the second statement we suppose $C = \delta(q, x)$ is a cut and $r \xrightarrow{y} q$ for some state $r \in C$. Then:

$$\delta(q, xyx) \;=\; \delta(C, yx) \;\supseteq\; \delta(r, yx) \;=\; \delta(\,\delta(r, y),\; x) \;\supseteq\; \delta(q, x) \;=\; C$$

is a cut that subsumes R. Hence, $C = \delta(q, xyx)$ by the first statement. □

As a consequence of Lemma 4.4, we get that if $C \subseteq Q$ such that $\mathcal{L}_\omega(q) \cap \mathcal{L}_\omega(p) = \varnothing$ for all states $q, p \in C$ with $q \neq p$ then:

$$C \text{ is a cut} \quad \text{iff} \quad \delta(C, y) \neq \varnothing \text{ for all } y \in \Sigma^*$$

Corollary 4.24 (Pure cuts in strongly connected UBA; see first part of Lemma 4.27)**.** *Let $\mathcal{U}$ be a strongly connected UBA with at least one final state. Then for each state q and each finite word x:*

$$\delta(q, x) \textit{ is a cut} \quad \textit{iff} \quad \delta(q, xy) \neq \varnothing \textit{ for all } y \in \Sigma^*$$

Proof. The languages of the states in $\delta(q, x)$ are pairwise disjoint by the unambiguity of $\mathcal{U}$. Hence, $\delta(q, x)$ is a (pure) cut if and only if $\mathcal{U}[\delta(q, x)]$ is almost universal. By Lemma 4.4, this is equivalent to the statement that $\delta(q, xy) \neq \varnothing$ for all $y \in \Sigma^*$. □

The following lemma yields that for positive, strongly connected UBA, the pure cuts constitute a non-trap BSCC in the deterministic automaton $\mathcal{U}_{\text{det}}$ obtained by applying the standard powerset construction.

Lemma 4.25 (See second part of Lemma 4.27)**.** *Suppose* $\Pr(\mathcal{L}_\omega(\mathcal{U})) > 0$ *and* $\mathcal{U}$ *is strongly connected. Then, for each cut* $C \subseteq Q$ *the following statements are equivalent:*

(1) C *is pure, i.e.,* $C = \delta(q, x)$ *for some state* $q \in Q$ *and some word* $x \in \Sigma^*$

(2) for each state $p \in Q$ *there exists a word* $z \in \Sigma^*$ *such that* $C = \delta(p, z)$

(3) C *is reachable from any other cut, i.e., if* C' *is a cut, then there exists a finite word* y *with* $C = \delta(C', y)$*.*

Proof. Obviously, (1) is a consequence of (2). Let us prove the implication (1) $\Longrightarrow$ (2). Suppose $\delta(q, x)$ is a cut and let $p \in Q$ be an arbitrary state. Pick some finite word y with $p \xrightarrow{y} q$. Then, $\delta(p, yx) \supseteq \delta(q, x)$. We get $\delta(p, yx) = \delta(q, x)$ by the second statement of Fact 4.23.

We now show the implication (2) $\Longrightarrow$ (3). Let C' be a cut and $p \in C'$. By assumption (2), there is some word z such that $C = \delta(p, z)$. Then, $\delta(C', z)$ is a cut as well and we have:

$$C \;=\; \delta(p, z) \;\subseteq\; \delta(C', z)$$

and therefore $C = \delta(C', z)$ by the first statement of Fact 4.23.

For the implication (3) $\Longrightarrow$ (1) we pick a cut C' of the form $C' = \delta(p, z)$ for some state $p \in C$ and word z. (Such a cut exists by the second statement in Fact 4.22.) By assumption (3) there is a word y with $C = \delta(C', y)$. But then $C = \delta(p, yz)$.

□

The above lemma shows that if $\mathcal{U}$ is positive, then $\mathcal{U}_{\text{det}}$ has exactly one non-trap BSCC consisting of the cuts of $\mathcal{U}$ that are reachable from some resp. all singleton(s). More precisely, if $\Pr(\mathcal{L}_\omega(\mathcal{U})) > 0$ and $\mathcal{U}$ is strongly connected, then the cuts of $\mathcal{U}$ that are reachable from some singleton constitute a BSCC $\mathcal{C}$ of $\mathcal{U}_{\text{det}}$. This is the only non-trap BSCC of $\mathcal{U}_{\text{det}}$ and $\mathcal{C}$ is reachable from each cut. Additionally, the trap BSCC $\{\varnothing\}$ is reachable if there are states q in $\mathcal{U}$ with $\Pr(\mathcal{L}_\omega(q)) < 1$.

Remark 4.26. *As the results above show: there is no cut* C *that is reachable from some singleton and that is not contained in the non-trap BSCC* $\mathcal{C}$*. However, there might be cuts outside* $\mathcal{C}$*. For example, let* $\mathcal{U} = (Q, \{a, b\}, \delta, Q_0, \operatorname{Inf}(F))$ *where*

$$Q \;=\; \{q_a, q_b, p_a, p_b\}, \qquad Q_0 \;=\; \{q_a, p_b\}, \qquad \text{and} \qquad F \;=\; \{q_a\}.$$

The transition function δ *is given by:*

$$\begin{array}{lcllcl} \delta(q_a, a) & = & \{q_a, q_b\} & \delta(p_a, a) & = & \{p_a, p_b\} \\ \delta(q_b, b) & = & \{p_a, p_b\} & \delta(p_b, b) & = & \{q_a, q_b\} \end{array}$$

and $\delta(\cdot) = \varnothing$ *in all remaining cases. The unambiguity of* $\mathcal{U}$ *is clear since the switch between the q- and p-states are deterministic and since*

$$\begin{aligned} \mathcal{L}_\omega(q_a) \;=\; \mathcal{L}_\omega(p_a) \;&=\; \{\, aw \;:\; w \in \{a, b\}^\omega \,\}, \\ \mathcal{L}_\omega(q_b) \;=\; \mathcal{L}_\omega(p_b) \;&=\; \{\, bw \;:\; w \in \{a, b\}^\omega \,\}. \end{aligned}$$

Thus, $\mathcal{U}$ is universal. The sets $\{q_a, q_b\}$, $\{p_a, p_b\}$ constitute the non-trap BSCC consisting of the cuts that are reachable from the four singletons. The set $C = \{q_a, p_b\}$ is a cut too, but C is not reachable from any singleton. However, $\delta(C, a) = \delta(C, b) = \{q_a, q_b\}$.

Corollary 4.24 and Lemma 4.25 are summarized in Lemma 4.27.

Lemma 4.27 (Characterization of pure cuts)**.** *Let $\mathcal{U}$ be a strongly connected UBA. For all $q \in Q$ and $z \in \Sigma^*$ we have: $\delta(q, z)$ is a cut iff $\delta(q, zy) \neq \varnothing$ for each word $y \in \Sigma^*$. Furthermore, if $\mathcal{U}$ is positive, then for each cut C:*

$$\begin{aligned} & C \text{ is pure, i.e., } C = \delta(q, z) \text{ for some state-word pair } (q, z) \in Q \times \Sigma^* \\ \textit{iff} \quad & \text{for each state } q \in Q \text{ there is some word } z \in \Sigma^* \text{ with } C = \delta(q, z) \\ \textit{iff} \quad & \text{for each cut } C' \text{ there is some word } y \in \Sigma^* \text{ with } C = \delta(C', y) \end{aligned}$$

By Lemma 4.2 and Lemma 4.27 we get:

Corollary 4.28. *If $\mathcal{U}$ is a strongly connected UBA, then*

$$\Pr(\mathcal{L}_\omega(\mathcal{U})) > 0 \text{ iff } \mathcal{U} \text{ has a pure cut.}$$

For the rest of Section 4.1.3, we suppose that $\mathcal{U}$ is positive and strongly connected. The second part of Lemma 4.27 yields that the pure cuts constitute a bottom strongly connected component of the automaton obtained from $\mathcal{U}$ using the standard powerset construction. The goal is now to design an efficient (polynomially time-bounded) algorithm for the generation of a pure cut. For this, we observe that if $q, p \in Q$, $q \neq p$, then $\{q, p\} \subseteq C$ for some pure cut C iff there exists a word y such that $\{q, p\} \subseteq \delta(q, y)$.

Lemma 4.29. *Suppose $\Pr(\mathcal{L}_\omega(\mathcal{U})) > 0$ and $\mathcal{U}$ is strongly connected. Let q, p be states in $\mathcal{U}$ with $q \neq p$. Then:*

$$\begin{aligned} & \{q, p\} \subseteq C \text{ for some pure cut } C \\ \textit{iff} \quad & \text{there is some finite word } z \text{ with } q \xrightarrow{z} q \xrightarrow{z} p. \end{aligned}$$

Proof. Let C be a pure cut that contains q and p. By Lemma 4.25 there is a word z with $C = \delta(q, z)$. Then, $C = \delta(C, z)$ and $\delta(p, z) = \varnothing$ by the unambiguity of $\mathcal{U}$ (see Fact 4.13). But then $q \xrightarrow{z} q \xrightarrow{z} p$. Vice versa, $q \xrightarrow{z} q \xrightarrow{z} p$ implies $\{q, p\} \subseteq \delta(C, z)$ for each cut C with $q \in C$.

□

Remark 4.30. *Lemma 4.29 yields that for all states q and p of a strongly connected UBA $\mathcal{U}$:*

- *If there is no word z with $q \xrightarrow{z} q \xrightarrow{z} p$, then there is no cut C with $\{q, p\} \subseteq C$.*
- *If $\Pr(\mathcal{L}_\omega(\mathcal{U})) > 0$ and $q \xrightarrow{z} q \xrightarrow{z} p$, then there is a pure cut C that contains q and p.*

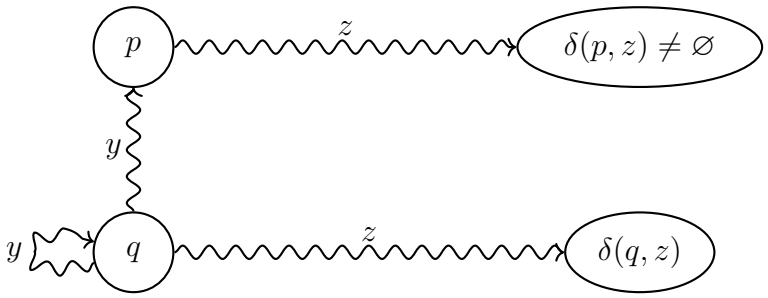


Figure 4.8: An extension $y \in \Sigma^*$ for a state-word pair $(q, z) \in Q \times \Sigma^*$. (Figure taken from [Kie17]).

The existence of some finite word z with $q \xrightarrow{z} q \xrightarrow{z} p$ can be checked efficiently using standard algorithms for NFA. Note that the first case applies (i.e., no such word z exists) if and only if the accepted languages of the NFA $\mathcal{B}_{q,q} = (Q, \Sigma, \delta, \{q\}, \mathrm{Reach}\,(\{q\}))$ and $\mathcal{B}_{q,p} = (Q, \Sigma, \delta, \{q\}, \mathrm{Reach}\,(\{p\}))$ are disjoint. The latter can be checked by running an emptiness check to the product-NFA of $\mathcal{B}_{q,q}$ and $\mathcal{B}_{q,p}$. If the languages of $\mathcal{B}_{q,q}$ and $\mathcal{B}_{q,p}$ are not disjoint, then we can generate a finite word z of length at most $|Q|^2$ such that $q \xrightarrow{z} q \xrightarrow{z} p$ by searching an accepted word of the product of $\mathcal{B}_{q,q}$ and $\mathcal{B}_{q,p}$.

Definition 4.31 (Extension)**.** *A word $y \in \Sigma^*$ is an extension for a state-word pair $(q, z) \in Q \times \Sigma^*$ iff there exists a state $p \in Q$ such that $q \neq p$, $\delta(p, z) \neq \varnothing$ and $\{q, p\} \subseteq \delta(q, y)$.*

The scheme for an extension is depicted in Figure 4.8. Occasionally, we refer to the pair (p, y) as an extension of (q, z). It is easy to see that if y is an extension of (q, z), then $\delta(q, yz)$ is a proper superset of $\delta(q, z)$ (see Lemma 4.32). Furthermore, for all state-word pairs $(q, z) \in Q \times \Sigma^*$ (see Lemma 4.33):

$$\delta(q, z) \text{ is a cut} \quad \text{iff} \quad \text{there is no extension for } (q, z)$$

These observations lead to the following algorithm for the construction of a pure cut. We pick an arbitrary state q in the UBA and start with the empty word $z_0 = \varepsilon$. The algorithm iteratively seeks for an extension for the state-word pair (q, z_i). If an extension y_i for (q, z_i) has been found, then we switch to the word $z_{i+1} = y_i z_i$. If no extension exists, then (q, z_i) is a pure cut. In this way, the algorithm generates an increasing sequence of subsets of Q,

$$\delta(q, z_0) \subsetneq \delta(q, z_1) \subsetneq \delta(q, z_2) \subsetneq \ldots \subsetneq \delta(q, z_k),$$

which terminates after at most $|Q|$ steps and yields a pure cut $\delta(q, z_k)$.

Lemma 4.32. *Let $\mathcal{U}$ be a (possibly non-positive, possibly not strongly connected) UBA, and let q, p be states in $\mathcal{U}$ and y, z finite words. If y is an extension of (q, z), then $\delta(q, yz)$ is a proper superset of $\delta(q, z)$.*

Proof. Since $\{q,p\} \subseteq \delta(q,y)$ we have:

$$\delta(q,z) \subseteq \;=\; \delta(q,z) \cup \delta(p,z) \;=\; \delta\big(\{q,p\},z\big) \;\subseteq\; \delta\big(\delta(q,y),z\big) \;=\; \delta(q,yz)$$

In particular, $\delta(q,z) \subseteq \delta(q,yz)$. The set $\delta(p,z)$ is non-empty (by the definition of extensions). Let r be an arbitrary state in $\delta(p,z)$. Then, $r \notin \delta(q,z)$, since otherwise the word yz would have two runs from q to r:

$$q \xrightarrow{y} q \xrightarrow{z} r \quad \text{and} \quad q \xrightarrow{y} p \xrightarrow{z} r,$$

which is impossible by the unambiguity of $\mathcal{U}$ (see Fact 4.13).

□

Lemma 4.33. *Let $\mathcal{U}$ be a positive, strongly connected UBA. Then for each state-word pair $(q,z) \in Q \times \Sigma^*$:*

$$\delta(q,z) \textit{ is a cut} \quad \textit{iff} \quad \textit{the pair } (q,z) \textit{ has no extension}$$

Proof. "$\Longrightarrow$": Let $\delta(q,z)$ be a cut and suppose by contradiction that there exists an extension y for (q,z). Let $p \in Q$ such that $q \neq p$, $\delta(p,z) \neq \varnothing$ and $\{q,p\} \subseteq \delta(q,y)$. The set $\delta(q,yz) \setminus \delta(q,z)$ is non-empty (Lemma 4.32). As $\Pr(\,\mathcal{L}_\omega(r)\,) > 0$ for all states r, we get:

$$\Pr\big(\,\mathcal{L}_\omega(\delta(q,yz))\,\big) \;=\; \Pr\big(\,\mathcal{L}_\omega(\delta(q,z))\,\big) \;+\; \sum_{\substack{r \,\in\, \delta(q,\,yz) \\ r \,\notin\, \delta(q,\,z)}} \Pr\big(\,\mathcal{L}_\omega(r)\,\big) \;>\; \Pr\big(\,\mathcal{L}_\omega(\delta(q,z))\,\big)$$

Hence, $\Pr(\mathcal{L}_\omega(\delta(q,z)) < 1$. But then $\delta(q,z)$ cannot be a cut as $\Pr(\mathcal{L}_\omega(R)) = 1$ for all cuts R. Contradiction.

"$\Longleftarrow$": Suppose now that (q,z) has no extension. We have to show that $C = \delta(q,z)$ is a cut.

We first observe that there is some cut R that contains C. For this, we may pick any cut R' with $q \in R'$. Then, $R = \delta(R',z)$ is a cut with $C \subseteq R$. For each state $p \in R \setminus \{q\}$, there is some finite word y with $q \xrightarrow{y} q \xrightarrow{y} p$ (see Lemma 4.29). Since there exists no extension for (q,z), we have $\delta(p,z) = \varnothing$ for all states $p \in R \setminus C$. This yields:

$$C \;=\; \delta(q,z) \;=\; \delta(C,z) \;=\; \delta(R,z)$$

By Fact 4.22, as R is a cut, $\delta(R,z)$ is a cut as well and hence so is C.

□

It remains to explain an efficient realization of the search for an extension of the state-word pairs (q,z_i). The idea is to store the sets $Q_i[p] = \delta(p,z_i)$ for all states p. The sets $Q_i[p]$ can be computed iteratively by:

$$Q_0[p] = \{p\} \quad \text{and} \quad Q_{i+1}[p] \;=\; \bigcup_{r \in \delta(p,\,y_i)} Q_i[r]$$

To check whether (q, z_i) has an extension we apply standard techniques for the intersection problem for the languages $H_{q,q} = \{y \in \Sigma^* : q \in \delta(q, y)\}$ and $H_{q,F_i} = \{y \in \Sigma^* : \delta(q, y) \cap F_i \neq \varnothing\}$ where $F_i = \{p \in Q \setminus \{q\} : Q_i[p] \neq \varnothing\}$. Then, for each word $y \in \Sigma^*$ we have: $y \in H_{q,q} \cap H_{q,F_i}$ if and only if y is an extension of (q, z_i). The languages $H_{q,q}$ and H_{q,F_i} are recognized by the NFA $\mathcal{U}_{q,q} = (Q, \Sigma, \delta, q, \mathrm{Reach}\,(q))$ and $\mathcal{U}_{q,F_i} = (Q, \Sigma, \delta, q, \mathrm{Reach}\,(F_i))$. Thus, to check the existence of an extension and to compute an extension y (if existent) where the word y has length at most $|Q|^2$, we may run an emptiness check for the product-NFA $\mathcal{U}[q, q] \otimes \mathcal{U}[q, F_i]$. We conclude:

Corollary 4.34. *Given a positive, strongly connected UBA $\mathcal{U}$, a pure cut can be computed in time polynomial in the word-length of $\mathcal{U}$.*

Computing the measure of positive, strongly connected UBA.

We suppose that $\mathcal{U} = \big(Q, \Sigma, \delta, Q_0, \mathrm{Inf}\,(F)\big)$ is a positive, strongly connected UBA and C is a cut. (C might be a pure cut that has been computed by the techniques explained above. However, in Theorem 4.35 C can be any cut.) Consider the linear equation system of Lemma 4.17 with variables ζ_q for all states $q \in Q$ and add the constraint that the variables ζ_q for $q \in C$ sum up to 1.

Theorem 4.35. *Let $\mathcal{U}$ be a positive, strongly connected UBA and C a cut. Then, the probability vector $(\Pr(\mathcal{L}_\omega(q)))_{q \in Q}$ is the unique solution of the following linear equation system:*

$$(1) \qquad \zeta_q \;=\; \frac{1}{|\Sigma|} \cdot \sum_{\sigma \in \Sigma} \sum_{p \in \delta(q,\sigma)} \zeta_p \qquad \textit{for all states } q \in Q$$

$$(2) \qquad \sum_{q \in C} \zeta_q \;=\; 1$$

Proof. Let $n = |Q|$. Define a matrix $M \in [0,1]^{Q \times Q}$ by $M_{q,p} = |\{\sigma \in \Sigma : p \in \delta(q, \sigma)\}|/|\Sigma|$ for all $q, p \in Q$. Then, the n equations (1) can be written as $\zeta = M\zeta$, where $\zeta = (\zeta_q)_{q \in Q}$ is a vector of n variables. It is easy to see that the values $\zeta^*_q = \Pr(\mathcal{L}_\omega(q))$ for $q \in Q$ satisfy the equations (1). That is, defining $\zeta^* = (\zeta^*_q)_{q \in Q}$ we have $\zeta^* = M\zeta^*$. By the definition of a cut, those values also satisfy equation (2).

It remains to show uniqueness. We employ Perron-Frobenius theory as follows. Since $\zeta^* = M\zeta^*$, the vector ζ^* is an eigenvector of M with eigenvalue 1. By ζ^* being strictly positive (i.e., positive in all components), it follows from [BP79, Corollary 2.1.12] that $\rho = 1$ for the spectral radius ρ of M. Since $\mathcal{U}$ is strongly connected, matrix M is irreducible. By [BP79, Theorem 2.1.4 (b)] the spectral radius $\rho = 1$ is a simple eigenvalue of M, i.e., all solutions of $\zeta = M\zeta$ are scalar multiples of ζ^*. Among those multiples, only ζ^* satisfies equation (2). Uniqueness follows.

□

Together with the criterion of Lemma 4.17 to check whether a given strongly connected UBA is positive, we obtain a polynomially time-bounded computation scheme for the values $\Pr(\mathcal{L}_\omega(q))$ for the states q of a given strongly connected UBA.

Foundations of the eigenvalue algorithm. The first approach for the positivity check (see page 73) was based on a rank computation. Now we give a second approach for the positivity check and a possible way to calculate the vector $(\Pr(\mathcal{L}_\omega(q)))_{q\in Q}$ as well. It relies on an iterative approximation of an eigenvector.

Let $\mathcal{U}$ be a strongly connected UBA with at least one final state. Let M be the matrix from the proof of Theorem 4.35. Define $\overline{M} = (I + M)/2$ where I denotes the $Q \times Q$ identity matrix. Denote by $\vec{1} = (1)_{q\in Q}$ the column vector where all components are 1. Define $\vec{0}$ similarly. For $i \geq 0$ define $v(i) = \overline{M}^i \cdot \vec{1}$. Our algorithm is as follows. Exploiting the recurrence $v(i+1) = \overline{M} \cdot v(i)$ we compute the sequence $v(0), v(1), \ldots$ until we find an $i > 0$ with either $v(i+1) < v(i)$ (by this inequality we mean strict inequality in all components) or $v(i+1) \approx v(i)$. In the first case we conclude that $\Pr(\mathcal{L}_\omega(\mathcal{U})) = 0$. In the second case we compute a cut C and multiply $v(i)$ by a scalar $c > 0$ so that $c \cdot \sum_{q\in C} v(i)_q = 1$, and conclude that $(\Pr(\mathcal{L}_\omega(q)))_{q\in Q} \approx c \cdot v(i)$. This algorithm is justified by the following two lemmas.

Lemma 4.36. *We have* $\Pr(\mathcal{L}_\omega(\mathcal{U})) = 0$ *if and only if there is* $i \geq 0$ *with* $v(i+1) < v(i)$.

Lemma 4.37. *If* $\Pr(\mathcal{L}_\omega(\mathcal{U})) > 0$, *then* $v(\infty) := \lim_{i\to\infty} v(i) > \vec{0}$ *exists, and* $Mv(\infty) = v(\infty)$, *and* $v(\infty)$ *is a scalar multiple of* $(\Pr(\mathcal{L}_\omega(q)))_{q\in Q}$.

For the proofs we need the following two auxiliary lemmas:

Lemma 4.38. *Let* $\rho > 0$ *denote the spectral radius of* $\overline{M}$. *Then the matrix limit* $\lim_{i\to\infty} \left(\overline{M}/\rho\right)^i$ *exists and is strictly positive in all entries.*

Proof. Since M is irreducible, $\overline{M}^{|Q|}$ is strictly positive (in all entries). Then it follows from [HJ13, Theorem 8.2.7] that the matrix limit

$$\lim_{i\to\infty} \left(\overline{M}/\rho\right)^i = \lim_{i\to\infty} \left(\left(\overline{M}/\rho\right)^{|Q|}\right)^i$$

exists and is strictly positive. □

Lemma 4.39. *For any* $v \in \mathbb{C}^Q$ *and any* $c \in \mathbb{C}$ *we have* $Mv = cv$ *if and only if* $\overline{M}v = \frac{1+c}{2}v$. *In particular,* M *and* $\overline{M}$ *have the same eigenvectors with eigenvalue* 1.

The proof of Lemma 4.39 is obvious. We prove now Lemma 4.36 and 4.37.

Proof of Lemma 4.36. Let $i \geq 0$ with $v(i+1) < v(i)$. With [BP79, Theorem 2.1.11] it follows that the spectral radius of $\overline{M}$ is smaller than 1, hence by Lemma 4.39 the spectral radius of M is smaller than 1 as well. By Lemma 4.18 it follows that $\Pr(\mathcal{L}_\omega(\mathcal{U})) = 0$.

For the converse, let $\Pr(\mathcal{L}_\omega(\mathcal{U})) = 0$. By Lemma 4.18, the spectral radius of M is smaller than 1. Let ρ denote the spectral radius of $\overline{M}$. By Lemma 4.39, we have $\rho < 1$. If $\rho = 0$, then $\overline{M}$ is the zero matrix and we have $v(1) = \vec{0} < \vec{1} = v(0)$. Let $\rho > 0$. It follows from Lemma 4.38 that there is $i \geq 0$ such that $\rho \left(\overline{M}/\rho\right)^{i+1} < \left(\overline{M}/\rho\right)^i$ (with the inequality strict in all components). Hence, $\overline{M}^{i+1} < \overline{M}^i$ and $v(i+1) = \overline{M}^{i+1} \cdot \vec{1} < \overline{M}^i \cdot \vec{1} = v(i)$.

□

Proof of Lemma 4.37. Let $\Pr(\mathcal{L}_\omega(\mathcal{U})) > 0$. Then, by Lemma 4.18, the spectral radius of M is 1. So, with Lemma 4.39 the spectral radius of $\overline{M}$ is 1. By Lemma 4.38 the limit $v(\infty) = \lim_{i\to\infty} \overline{M}^i \vec{1}$ exists and is positive. From the definition of $v(\infty)$ we have $\overline{M}v(\infty) = v(\infty)$. By Lemma 4.39 also $Mv(\infty) = v(\infty)$. So $v(\infty)$ solves equation (1) from Theorem 4.35. There is a scalar $c > 0$ so that $cv(\infty)$ satisfies both equations (1) and (2) from Theorem 4.35. By the uniqueness statement of Theorem 4.35 it follows that $cv(\infty) = (\Pr(\mathcal{L}_\omega(q)))_{q\in Q}$.

□

The next section shows how to lift these results for arbitrary UBA.

4.1.4 Computing the measure of arbitrary UBA

In what follows, let $\mathcal{U} = (Q, \Sigma, \delta, Q_0, \mathrm{Inf}(F))$ be a (possibly not strongly connected) UBA and $\mathcal{C}$ a strongly connected component (SCC) of $\mathcal{U}$. $\mathcal{C}$ is called non-trivial if $\mathcal{C}$ viewed as a direct graph contains at least one edge, i.e., if $\mathcal{C}$ is cyclic. Recall that $\mathcal{C}$ is called bottom if $\delta(q, a) \subseteq \mathcal{C}$ for all $q \in \mathcal{C}$ and all $a \in \Sigma$. If $\mathcal{C}$ is a non-trivial SCC of $\mathcal{U}$ and $p \in \mathcal{C}$, then the sub-NBA

$$\mathcal{U}\big|_{\mathcal{C},p} \quad = \quad \big(\mathcal{C}, \Sigma, \delta|_{\mathcal{C}}, \{p\}, \mathrm{Inf}(\mathcal{C} \cap F)\big)$$

of $\mathcal{U}$ with state space $\mathcal{C}$, initial state p and the transition function $\delta|_{\mathcal{C}}$ given by $\delta|_{\mathcal{C}}(q, a) = \delta(q, a) \cap \mathcal{C}$ is strongly connected and unambiguous. Let L_p be the accepted language, i.e., $L_p = \mathcal{L}_\omega(\mathcal{U}\big|_{\mathcal{C},p})$. The values $\Pr(L_p)$, $p \in \mathcal{C}$, can be computed using the techniques for strongly connected UBA presented in Section 4.1.3. A non-trivial SCC $\mathcal{C}$ is said to be positive if $\Pr(L_p) > 0$ for all/some state(s) p in $\mathcal{C}$.

We perform the following preprocessing. As before, for any $p \in Q$ we write $\mathcal{L}_\omega(p)$ for $\mathcal{L}_\omega(\mathcal{U}[p])$, and call p zero if $\Pr(\mathcal{L}_\omega(p)) = 0$. First we remove all states that are not reachable from any initial state. Then we run standard graph algorithms to compute the directed acyclic graph (DAG) of SCCs of $\mathcal{U}$. By processing the DAG bottom-up we can remove all zero states by running the following loop: If all BSCCs are marked (initially, all SCCs are unmarked), then exit the loop; otherwise pick an unmarked BSCC $\mathcal{C}$.

- If $\mathcal{C}$ is trivial or does not contain any final state, then we remove it: more precisely, we remove it from the DAG of SCCs, and we modify $\mathcal{U}$ by deleting all transitions $p \xrightarrow{\sigma} q$ where $q \in \mathcal{C}$.
- Otherwise, $\mathcal{C}$ is a non-trivial BSCC with at least one final state. We check whether $\mathcal{C}$ is positive by applying the techniques of Section 4.1.3. If it is positive, we mark it; otherwise we remove it as described above.

Note that this loop does not change $\Pr(\mathcal{L}_\omega(p))$ for any state p.

Let Q_{BSCC} denote the set of states in $\mathcal{U}$ that belong to some BSCC. The values $\Pr(\mathcal{L}_\omega(p))$ for the states $p \in Q_{BSCC}$ can be computed using the techniques of

Section 4.1.3. The remaining task is to compute the values $\Pr(\mathcal{L}_\omega(q))$ for the states $q \in Q \setminus Q_{BSCC}$. For $q \in Q \setminus Q_{BSCC}$, let $\beta_q = 0$ if $\delta(q,\sigma) \cap Q_{BSCC} = \varnothing$ for all $\sigma \in \Sigma$. Otherwise:

$$\beta_q \;=\; \frac{1}{|\Sigma|} \cdot \sum_{\sigma \in \Sigma} \sum_{p \in \delta(q,\sigma) \cap Q_{BSCC}} \Pr(\mathcal{L}_\omega(p))$$

We now show that the probabilities $\Pr(\mathcal{L}_\omega(q))$ for $q \in Q \setminus Q_{BSCC}$ are computable by the linear equation system shown in Figure 4.9 with $|Q \setminus Q_{BSCC}|$ equations and variables ζ_q for $q \in Q \setminus Q_{BSCC}$.

$$\zeta_q \;=\; \frac{1}{|\Sigma|} \cdot \sum_{\sigma \in \Sigma} \sum_{\substack{r \,\in\, \delta(q,\sigma) \\ r \,\notin\, Q_{BSCC}}} \zeta_r \;+\; \beta_q \qquad \text{for } q \in Q \setminus Q_{BSCC}$$

Figure 4.9: Linear equation system for computing $\Pr(\mathcal{L}_\omega(q))$ in UBA.

Theorem 4.40. *If all BSCCs of $\mathcal{U}$ are non-trivial and positive, then the linear equation system in Figure 4.9 has a unique solution, namely $\zeta_q^* = \Pr(\mathcal{L}_\omega(q))$.*

Proof. We write $\overline{Q}$ for $Q \setminus Q_{BSCC}$ and define a matrix $M \in [0,1]^{\overline{Q} \times \overline{Q}}$ by

$$M_{q,p} \;=\; \frac{|\{\sigma \in \Sigma \,:\, p \in \delta(q,\sigma)\}|}{|\Sigma|} \quad \text{for all } q,p \in \overline{Q}.$$

Further, we define a vector $\beta \in [0,1]^{\overline{Q}}$ with $\beta = (\beta_q)_{q \in \overline{Q}}$. Then the equation system in Figure 4.9 can be written as

$$\zeta \;=\; M\zeta + \beta\,,$$

where $\zeta = (\zeta_q)_{q \in \overline{Q}}$ is a vector of variables. By reasoning similar to the beginning of the proof of Lemma 4.17 one can show that the values $\zeta_q^* = \Pr(\mathcal{L}_\omega(q))$ for $q \in \overline{Q}$ satisfy this equation system. That is, by defining $\zeta^* = (\zeta_q^*)_{q \in \overline{Q}}$ we have $\zeta^* = M\zeta^* + \beta$.

It remains to show uniqueness. Since β is non-negative,

$$\zeta^* \geqslant M\zeta^*$$

holds where the inequalities hold component-wise. As M is non-negative, it follows from monotonicity that we have

$$\zeta^* \;\geqslant\; M\zeta^* \;\geqslant\; M^2\zeta^* \;\geqslant\; \ldots \tag{4.4}$$

For $i \geqslant 1$, let Q_i be the set of states in $\overline{Q}$ that have a path in $\mathcal{U}$ of length i or shorter to a BSCC of $\mathcal{U}$. We prove by induction on i that for all $i \geq 1$:

$$\left(M^i\zeta^*\right)_q \;<\; \zeta_q^* \qquad \text{for all } q \in Q_i. \tag{4.5}$$

For $i = 1$, note that $\beta_q > 0$ for all $q \in Q_1$, hence $(M\zeta^*)_q < (M\zeta^* + \beta)_q = \zeta^*_q$ for all $q \in Q_1$. For the step of induction, let $q \in Q_{i+1}$ for $i \geqslant 1$. Then, there exist $\sigma \in \Sigma$ and $p \in \delta(q,\sigma) \cap Q_i$. Hence, $M_{q,p} > 0$. By induction hypothesis we have $(M^i\zeta^*)_p < \zeta^*_p$. This yields:

$$\begin{aligned}
\left(M^{i+1}\zeta^*\right)_q &= \sum_{r\in\overline{Q}} M_{q,r}\left(M^i\zeta^*\right)_r \\
&= \underbrace{M_{q,p}}_{>0}\underbrace{\left(M^i\zeta^*\right)_p}_{<\zeta^*_p} + \sum_{r\in\overline{Q}\setminus\{p\}} M_{q,r}\big(\underbrace{M^i\zeta^*}_{\leqslant\zeta^* \text{ by (4.4)}}\big)_r \\
&< M_{q,p}\cdot\zeta^*_p + \sum_{r\in\overline{Q}\setminus\{p\}} M_{q,r}\zeta^* \\
&= (M\zeta^*)_q \\
&\leqslant \zeta^*_q \qquad \text{by (4.4).}
\end{aligned}$$

This shows (4.5). Since $Q_{|\overline{Q}|} = \overline{Q}$ it follows

$$M^{|\overline{Q}|}\zeta^* \;<\; \zeta^*\,, \tag{4.6}$$

where the inequality is strict in all components. So there is $0 < c < 1$ with $M^{|\overline{Q}|}\zeta^* \leqslant c\zeta^*$. By induction, it follows $M^{|\overline{Q}|\cdot i}\zeta^* \leqslant c^i\zeta^*$ for all $i \geqslant 0$. Thus, $\lim_{i\to\infty} M^i\zeta^* = 0$, where 0 denotes the zero vector. By (4.6) the vector ζ^* is strictly positive. It follows that $\lim_{i\to\infty} M^i = [0]$, where $[0]$ denotes the zero matrix.

Let $x \in \mathbb{R}^{\overline{Q}}$ be an arbitrary solution of the equation system in Figure 4.9, i.e., $x = Mx + \beta$. Since $\zeta^* = M\zeta^* + \beta$, subtracting the two solutions yields

$$x-\zeta^* = M(x-\zeta^*) \;=\; M^2(x-\zeta^*) \;=\; \ldots \;=\; \lim_{i\to\infty} M^i(x-\zeta^*) \;=\; [0](x-\zeta^*) = 0\,,$$

where 0 denotes the zero vector. Hence, $x = \zeta^*$, which proves uniqueness of the solution.

□

Theorem 4.40 yields that the value $\Pr(\mathcal{L}_\omega(\mathcal{U}))$ for a given UBA $\mathcal{U}$ is computable in polynomial time.

Remark 4.41. *For the special case where $\delta(q,\sigma) = \{q\}$ for all $q \in F$ and $\sigma \in \Sigma$, the language of $\mathcal{U}$ is a cosafety property and under the assumption that all BSCCs are non-trivial and positive, $\Pr(\mathcal{L}_\omega(q)) = 1$ if $q \in F = Q_{BSCC}$. In this case, the linear equation system in Theorem 4.40 coincides with the linear equation system presented in [BLW13b] for computing the probability measure of the language of $\mathcal{U}$ viewed as a UFA.*

Remark 4.42. *As a consequence of our results, the positivity problem ("does $\Pr(\mathcal{L}_\omega(\mathcal{U})) > 0$ hold?") and the almost universality problem ("does $\Pr(\mathcal{L}_\omega(\mathcal{U})) = 1$*

hold?") for UBA are solvable in polynomial time. This should be contrasted with the standard (non-probabilistic) semantics of UBA and the corresponding results for NBA. The non-emptiness problem for UBA is in P (this already holds for NBA), while the complexity-theoretic status of the universality problem for UBA is a long-standing open problem. For standard NBA, it is well-known that the non-emptiness problem is in P and the universality problem is PSPACE-complete. However, the picture changes when switching to NBA with the probabilistic semantics as both the positivity problem and the almost universality problem for NBA are PSPACE-complete, even for strongly connected NBA.

Theorem 4.43 (see also [Var85; CY95])**.** *The positivity and the almost universality problem for strongly connected NBA are PSPACE-complete.*

Proof. Membership in PSPACE follows from the results of [Var85; CY95]. PSPACE-hardness of the positivity and almost universality problem for strongly connected NBA can be established using a polynomial reduction from the universality problem for non-deterministic finite automata (NFA) where all states are final. The latter problem is known to be PSPACE-complete [KRS09].

Let $\mathcal{B} = (Q, \Gamma, \delta_{\mathcal{B}}, Q_0, \mathrm{Reach}(Q))$ be an NFA. We can safely assume that Q_0 is non-empty and all states are reachable from Q_0. We define an NBA $\mathcal{A} = (Q, \Sigma, \delta_{\mathcal{A}}, Q_0, \mathrm{Inf}(Q))$ over the alphabet $\Sigma = \Gamma \cup \{\#\}$ as follows. If $q \in Q$ and $a \in \Gamma$, then $\delta_{\mathcal{A}}(q, a) = \delta_{\mathcal{B}}(q, a)$ and $\delta_{\mathcal{A}}(q, \#) = Q_0$. Clearly, $\mathcal{A}$ is strongly connected. Furthermore, $\delta_{\mathcal{A}}(R, \#) = Q_0$ for all non-empty subsets R of Q and $\delta_{\mathcal{A}}(q, x\#) = Q_0$ for all states q and all words $x \in \Sigma^*$ where $\delta_{\mathcal{A}}(q, x)$ is non-empty.

$$\begin{aligned} \mathcal{B} \text{ is universal} \quad & \text{iff} \quad \mathcal{L}_{\mathrm{fin}}(\mathcal{B}) = \Gamma^* \\ & \text{iff} \quad \delta_{\mathcal{B}}(Q_0, y) \neq \varnothing \text{ for all } y \in \Gamma^* \\ & \text{iff} \quad \delta_{\mathcal{A}}(Q_0, x) \neq \varnothing \text{ for all } x \in \Sigma^* \end{aligned}$$

By Lemma 4.4 (see Section 4.1.3), $\mathcal{B}$ is universal iff $\Pr(\mathcal{L}_\omega(\mathcal{A})) = 1$. This yields the PSPACE-hardness of the almost universality problem for strongly connected NBA. Moreover, all singletons $\{q\}$ can reach Q_0, and if there is a non-trap BSCC $\mathcal{C}$ of $\mathcal{A}_{\mathrm{det}}$, then $Q_0 \in \mathcal{C}$. Using Corollary 4.9, we obtain:

$$\begin{aligned} \mathcal{B} \text{ is universal} \quad & \text{iff} \quad Q_0 \text{ is contained in some non-trap BSCC of } \mathcal{A}_{\mathrm{det}} \\ & \text{iff} \quad \Pr(\mathcal{L}_\omega(\mathcal{A})) > 0 \end{aligned}$$

This yields the PSPACE-hardness of the positivity problem for strongly connected NBA.

□

4.1.5 Probabilistic model checking of Markov chains against UBA

To complete the proof of Theorem 4.1, we show how the results of the previous section can be adapted to compute the value $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{U}))$ for a Markov chain $\mathcal{M} = (S, P, \iota)$

and a UBA $\mathcal{U} = \big(Q, \Sigma, \delta, Q_0, \mathrm{Inf}\,(F)\big)$ with alphabet $\Sigma = S$.[4]

For an NBA $\mathcal{A}$ over the alphabet S, we write $\Pr_{\mathcal{M}[s]}(\mathcal{A})$ for $\Pr_{\mathcal{M}[s]}(\Pi)$ where Π denotes the set of infinite paths $s_0\, s_1\, s_2\, s_3 \ldots$ in the Markov chain $\mathcal{M}$ such that $s_0 = s$ and $s_1\, s_2\, s_3 \ldots \in \mathcal{L}_\omega(\mathcal{A})$. Thus, $\Pr_{\mathcal{M}[s]}(\mathcal{A}) = \Pr_{\mathcal{M}[s]}(L)$ where $L = \{sw : w \in \mathcal{L}_\omega(\mathcal{A})\}$. In contrast, $\Pr_{\mathcal{M}[s]}(\mathcal{L}_\omega(\mathcal{A}))$ denotes the probability of the set of infinite paths w in $\mathcal{M}$ that start in s and are accepted by $\mathcal{A}$. Thus, $\Pr_{\mathcal{M}[s]}(\mathcal{L}_\omega(\mathcal{A}))$ and $\Pr_{\mathcal{M}[s]}(\mathcal{A})$ might be different.

As a first step, we build a UBA $\mathcal{P} = \mathcal{M} \otimes \mathcal{U}$ that arises from the synchronous product of the UBA $\mathcal{U}$ with the underlying graph of the Markov chain $\mathcal{M}$. Formally, if $\mathcal{M} = (S, P, \iota)$ is a Markov chain $\mathcal{M}$ and $\mathcal{A} = \big(Q, S, \delta, Q_0, \mathrm{Inf}\,(F)\big)$ an NBA with the alphabet S, then

$$\mathcal{M} \otimes \mathcal{A} \;=\; \big(S \times Q, S, \Delta, Q_0', \mathrm{Inf}\,(S \times F)\big)$$

where

$$Q_0' \;=\; \big\{\, \langle s, q\rangle \in S \times Q \,:\, \iota(s) > 0,\ q \in \delta(Q_0, s) \,\big\}$$

and for $s, t \in S$ with $P(s,t) > 0$ and $q \in Q$:

$$\Delta(\langle s, q\rangle, t) \;=\; \big\{\, \langle t, p\rangle \,:\, p \in \delta(q, t) \,\big\}$$

If $P(s,t) = 0$, then $\Delta(\langle s, q\rangle, t) = \varnothing$. Given that $\mathcal{M}$ viewed as an automaton over the alphabet S behaves deterministically, the NBA $\mathcal{M} \otimes \mathcal{A}$ is unambiguous if $\mathcal{A}$ is a UBA.

To adapt the soundness proofs accordingly we need to "relativize" the probabilities for words in S^* according to the paths in $\mathcal{M}$. That is, we have to switch from the uniform probability measure Pr over the σ-algebra spanned by the cylinder sets of the finite words $\mathit{Cyl}(x) = \{xw : w \in \Sigma^\omega\}$ to the measures $\Pr_{\mathcal{M}[s]}$ induced by the states of $\mathcal{M}$.

We now illustrate how the proofs can be adapted by a few central statements.

Lemma 4.44 (cf. Lemma 4.2)**.** *Let $s \in S$ be a state of $\mathcal{M}$ and $L \subseteq S^\omega$ be an ω-regular language with $\Pr_{\mathcal{M}[s]}(L) > 0$. Then, there exists a finite path $x \in S^*$ starting in state s such that almost all extensions of x belong to L according to the measure $\Pr_{\mathcal{M}[s]}$, i.e.,*

$$\Pr_{\mathcal{M}[s]}\big\{\, w \in L \,:\, x \in \mathrm{Pref}(w) \,\big\} \;=\; \Pr_{\mathcal{M}[s]}\big(\, \mathit{Cyl}(x) \,\big)$$

where $\mathrm{Pref}(w)$ denotes the set of finite prefixes of w.

[4] In practice, e.g., when the UBA is obtained from an LTL formula, the alphabet of the UBA is often defined as $\Sigma = 2^{AP}$ over a set of atomic propositions AP and the Markov chain is equipped with a labeling function from states to the atomic propositions that hold in each state. Clearly, unambiguity w.r.t. the alphabet 2^{AP} implies unambiguity w.r.t. the alphabet S when switching from the original transition function $\delta : Q \times 2^{AP} \to 2^Q$ to the transition function $\delta_S : Q \times S \to 2^Q$ given by $\delta_S(q, s) = \delta(q, L(s))$, where $L : S \to 2^{AP}$ denotes the labeling function of $\mathcal{M}$.

Proof. The argument is fairly the same as in the proof of Lemma 4.2. We regard a deterministic automaton $\mathcal{D}$ for L and consider the product Markov chain $\mathcal{M} \otimes \mathcal{D}$. In this context, we consider $\mathcal{M}$ as a transition-labeled Markov chain where all outgoing transitions of state s are labeled with s. If $\Pr_{\mathcal{M}[s]}(L)$ is positive, then there is a finite path x from s such that the lifting of x from $\langle s, q_{init}\rangle$ in the product ends in a state that belongs to some BSCC where the acceptance condition of $\mathcal{D}$ holds. But then, $w \in L$ for almost all infinite paths w in $\mathcal{M}$ with $x \in \mathrm{Pref}(w)$.

□

Lemma 4.45 (Generalization of Lemma 4.4)**.** *For each NBA $\mathcal{A}$ over the alphabet S where $\mathcal{M} \otimes \mathcal{A}$ is strongly connected we have:*

$$\Pr_{\mathcal{M}}\big(\,\mathcal{L}_\omega(\mathcal{A})\,\big) \;=\; 1 - \Pr_{\mathcal{M}}\big\{\, w \in S^\omega \,:\, \delta(Q_0, x) = \varnothing \text{ for some } x \in \mathrm{Pref}(w)\,\big\}$$

In particular, $\Pr_{\mathcal{M}}(\,\mathcal{L}_\omega(\mathcal{A})\,) = 1$ *if and only if* $\delta(Q_0, x) \neq \varnothing$ *for all finite words* $x \in S^+$ *where x is a finite path in $\mathcal{M}$ starting in s.*

Proof. The arguments transfers from the proof of Lemma 4.4. Instead of Fact 4.6, we can rely on the original result of [CY95] for checking whether $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{A})) > 0$ for a given Markov chain $\mathcal{M}$. Note that the only reference to the probability measure is in the form "almost all words in L enjoy property XY" which simply means "the words in L not satisfying XY constitute a null set". At a few places we used $1/|\Sigma|^{|x|}$ as the measure of (the cylinder set spanned by) the finite word x as label of some path from state p to q. The value $1/|\Sigma|^{|x|}$ has to be replaced with $\Pr_s(\mathit{Cyl}(x))$ for the corresponding state s in $\mathcal{M}$. Note that if $\langle s, p\rangle \xrightarrow{x} \langle t, q\rangle$ in $\mathcal{M} \otimes \mathcal{A}$, then $\Pr_s(\mathit{Cyl}(x))$ is positive.

□

Our algorithm relies on the observation that

$$\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{U})) \;=\; \sum_{s \in S} \iota(s) \cdot \Pr_{\mathcal{M}[s]}(\,\mathcal{U}[\delta(Q_0, s)]\,)$$

for a Markov Chain $\mathcal{M}$ and UBA $\mathcal{U}$. As the languages of the UBA $\mathcal{U}[q]$ for $q \in \delta(Q_0, s)$ are pairwise distinct (by the unambiguity of $\mathcal{U}$), we have $\Pr_{\mathcal{M}[s]}(\mathcal{U}[\delta(Q_0, s)]) = \sum_{q \in \delta(Q_0, s)} \Pr_{\mathcal{M}[s]}(\mathcal{U}[q])$. Thus, the task is to compute the values $\Pr_{\mathcal{M}[s]}(\mathcal{U}[q])$ for $s \in S$ and $q \in Q$.

Let $\mathcal{P}[s, q]$ denote the UBA resulting from $\mathcal{P}$ by declaring $\langle s, q\rangle$ to be initial. It is easy to see that $\Pr_{\mathcal{M}[s]}(\,\mathcal{P}[s, q]\,) = \Pr_{\mathcal{M}[s]}(\,\mathcal{U}[q]\,)$ for all states $\langle s, q\rangle$ of $\mathcal{P}$, as the product construction only removes transitions in $\mathcal{U}$ that can not occur in the Markov chain. Our goal is thus to compute the values $\Pr_{\mathcal{M}[s]}(\mathcal{P}[s, q])$. For this, we remove all states $\langle s, q\rangle$ from $\mathcal{P}$ that can not reach a state in $S \times F$. Then, we determine the non-trivial SCCs of $\mathcal{P}$ and, for each such SCC $\mathcal{C}$, we analyze the sub-UBA $\mathcal{P}\big|_{\mathcal{C}}$ obtained by restricting to the states in $\mathcal{C}$. An SCC $\mathcal{C}$ of $\mathcal{P}$ is called positive if $\Pr_{\mathcal{M}[s]}(\mathcal{P}\big|_{\mathcal{C}}[s, q]) > 0$ for all/any $\langle s, q\rangle \in \mathcal{C}$. As in Section 4.1.4, we treat the SCCs in a bottom-up manner, starting with the BSCCs and removing them if they are

non-positive. Clearly, if a BSCC $\mathcal{C}$ of $\mathcal{P}$ does not contain a final state or is trivial, then $\mathcal{C}$ is not positive. Analogously to Lemma 4.17, we can check whether a non-trivial BSCC $\mathcal{C}$ in $\mathcal{P}$ containing at least one final state is positive by analyzing a linear equation system.

Lemma 4.46 (cf. Lemma 4.17)**.** *Let $\mathcal{C}$ be a BSCC of $\mathcal{P}$ and*

$$(*) \qquad \zeta_{s,q} \;=\; \sum_{t \in \mathrm{Post}(s)} \; \sum_{p \in \delta_{\mathcal{C}}(q,t)} P(s,t) \cdot \zeta_{t,p} \quad \text{for all } \langle s,q \rangle \in \mathcal{C}.$$

Then, the following statements are equivalent:

(1) $\mathcal{C}$ is positive,

(2) the linear equation system $(*)$ has a positive solution,

(3) the linear equation system $(*)$ has a non-zero solution.

Proof. The proof of Lemma 4.17 presented in Section 4.1.3 is directly applicable here as well by considering the $|\mathcal{C}| \times |\mathcal{C}|$-matrix M with

$$M_{\langle s,q \rangle, \langle t,p \rangle} \;=\; \begin{cases} P(s,t) & : \text{ if } p \in \delta(q,t) \\ 0 & : \text{ otherwise} \end{cases}$$

Note that the matrix M is non-negative and irreducible and the entry in the n-th power of M for state $\langle s,q \rangle$ and $\langle t,p \rangle$ is the probability with respect to $\Pr_{\mathcal{M}[s]}$ of all infinite paths $w = s_0\, s_1\, s_2 \ldots$ in $\mathcal{M}$ with $s_0 = s$ such that $p \in \Delta_{\mathcal{C}}(q, s_1 \ldots s_n)$. (Recall that $\Delta_{\mathcal{C}}$ denotes the transition relation of $\mathcal{P}$ restricted to $\mathcal{C}$.)

□

We now explain how to adapt the cut-based approach of Section 4.1.3 for computing the probabilities in a positive BSCC $\mathcal{C}$ of $\mathcal{P}$.

For $\langle s,q \rangle \in \mathcal{C}$ and $t \in S$, let $\Delta_{\mathcal{C}}(\langle s,q \rangle, t) = \Delta(\langle s,q \rangle, t) \cap \mathcal{C}$. A pure cut in $\mathcal{C}$ denotes a set $C \subseteq \mathcal{C}$ such that $\Pr_{\mathcal{M}[s]}(\mathcal{P}[C]) = 1$ and $C = \Delta_{\mathcal{C}}(\langle s,q \rangle, z)$ for some $\langle s,q \rangle \in \mathcal{C}$ and some finite word $z \in S^*$ such that $s\,z$ is a cycle in $\mathcal{M}$. (In particular, the last symbol of z is s, and therefore $C \subseteq \{\langle s,p \rangle \in \mathcal{C} : p \in Q\}$.)

Using Lemma 4.45 we can now adapt Corollary 4.24 and obtain:

Corollary 4.47 (Pure cuts in BSCCs of the product; cf. Corollary 4.24)**.** *Let $\mathcal{C}$ be a positive BSCC of $\mathcal{P}$ with at least one final state. Then, for each state $\langle s,q \rangle \in \mathcal{C}$ and each finite word x, the following two statements are equivalent:*

(a) $\{\langle s,p \rangle : p \in \delta_{\mathcal{C}}(q,x)\}$ is a pure cut.

(b) For each $y \in S^*$, there is some $p \in \delta_{\mathcal{C}}(q,x)$ such that $\Delta_{\mathcal{C}}(\langle s,p \rangle, y) \neq \varnothing$.

As before, let $\mathcal{C}$ be a positive BSCC of the product-UBA $\mathcal{P}$. Given a state $\langle s,q \rangle$ in $\mathcal{C}$ and a word $z \in S^*$, a word $y \in S^*$ is said to be an *extension* of $(\langle s,q \rangle, z)$ if the following two conditions hold:

(1) $s\,y$ is a cycle in the Markov chain $\mathcal{M}$

(2) there exists a state $p \in Q \setminus \{q\}$ in $\mathcal{U}$ such that $\Delta_{\mathcal{C}}(\langle s,p\rangle, z) \neq \varnothing$ and $\{\langle s,q\rangle, \langle s,p\rangle\} \subseteq \Delta_{\mathcal{C}}(\langle s,q\rangle, y)$.

Note that (1) implies, if $y = t_0\, t_1 \ldots t_m$, then $t_0 \in \mathrm{Post}(s)$ and $t_m = s$. In what follows, for $s,t \in S$, $q,p \in Q$ and $x \in S^*$, we often write

$$\langle s,q\rangle \ \xrightarrow{x}_{\mathcal{C}}\ \langle t,p\rangle$$

to indicate that $\langle t,p\rangle \in \Delta_{\mathcal{C}}(\langle s,q\rangle, x)$.

Lemma 4.48 (cf. Lemma 4.32)**.** *If y is an extension of $(\langle s,q\rangle, z)$, then $\Delta_{\mathcal{C}}(\langle s,q\rangle, yz)$ is a proper superset of $\Delta_{\mathcal{C}}(\langle s,q\rangle, z)$.*

Proof. Let $p \in Q \setminus \{q\}$ in $\mathcal{U}$ such that $\Delta_{\mathcal{C}}(\langle s,p\rangle, z) \neq \varnothing$ and $\{\langle s,q\rangle, \langle s,p\rangle\} \subseteq \Delta_{\mathcal{C}}(\langle s,q\rangle, y)$ (see condition (2)).

We first show that $\Delta_{\mathcal{C}}(\langle s,q\rangle, z) \subseteq \Delta_{\mathcal{C}}(\langle s,q\rangle, yz)$. For this, we pick a pair $\langle t,r\rangle \in \Delta_{\mathcal{C}}(\langle s,q\rangle, z)$. By condition (2) of extensions, we have $\langle s,q\rangle \in \Delta_{\mathcal{C}}(\langle s,q\rangle, z)$. Then:

$$\langle s,q\rangle \ \xrightarrow{y}_{\mathcal{C}}\ \langle s,q\rangle \ \xrightarrow{z}_{\mathcal{C}}\ \langle t,r\rangle$$

and therefore $\langle t,r\rangle \in \Delta_{\mathcal{C}}(\langle s,q\rangle, yz)$.

To show that the inclusion is strict we prove that $\Delta_{\mathcal{C}}(\langle s,p\rangle, z) \cap \Delta_{\mathcal{C}}(\langle s,q\rangle, z) = \varnothing$. We suppose by contradiction that $\langle t,r\rangle \in \Delta_{\mathcal{C}}(\langle s,p\rangle, z) \cap \Delta_{\mathcal{C}}(\langle s,q\rangle, z)$. Then:

$$\langle s,q\rangle \ \xrightarrow{y}_{\mathcal{C}}\ \langle s,q\rangle \ \xrightarrow{z}_{\mathcal{C}}\ \langle t,r\rangle \qquad \text{and} \qquad \langle s,q\rangle \ \xrightarrow{y}_{\mathcal{C}}\ \langle s,p\rangle \ \xrightarrow{z}_{\mathcal{C}}\ \langle t,r\rangle$$

But then $q \xrightarrow{y} q \xrightarrow{z} t$ and $q \xrightarrow{y} p \xrightarrow{z} t$ in $\mathcal{U}$. This is impossible by the unambiguity of $\mathcal{U}$.

□

Lemma 4.49 (cf. Lemma 4.33)**.** *As before, let $\mathcal{C}$ be a positive BSCC of $\mathcal{P}$. Then for each $\langle s,q\rangle \in \mathcal{C}$ and word $z \in S^+$ where the last symbol is s:*

$$\Delta_{\mathcal{C}}(\langle s,q\rangle, z) \text{ is a pure cut} \quad \textit{iff} \quad \text{the pair } (\langle s,q\rangle, z) \text{ has no extension}$$

Proof. The proof is fairly similar to the proof of Lemma 4.33.

"$\Longrightarrow$": Let $C = \Delta_{\mathcal{C}}(\langle s,q\rangle, z)$ be a pure cut. Suppose by contradiction that there exists an extension y for $(\langle s,q\rangle, z)$. Let $p \in Q$ such that $q \neq p$, $\Delta_{\mathcal{C}}(\langle s,p\rangle, z) \neq \varnothing$ and $\{\langle s,q\rangle, \langle s,p\rangle\} \subseteq \Delta_{\mathcal{C}}(\langle s,q\rangle, y)$. By Lemma 4.48, the set $\Delta_{\mathcal{C}}(\langle s,q\rangle, yz)$ is a proper superset of $\Delta(\langle s,q\rangle, z)$. As $\mathcal{C}$ is positive, we have $\Pr_u^{\mathcal{M}}(\,\mathcal{P}[u,r]\,) > 0$ for all $\langle u,r\rangle \in \mathcal{C}$. With an argument as in the proof of Lemma 4.33 we obtain:

$$\Pr_{\mathcal{M}[s]}\big(\,\mathcal{P}[\Delta_{\mathcal{C}}(\langle s,q\rangle, yz)]\,\big) \quad > \quad \Pr_{\mathcal{M}[s]}\big(\,\mathcal{P}[\Delta_{\mathcal{C}}(\langle s,q\rangle, z)]\,\big)$$

Hence, $\Pr_{\mathcal{M}[s]}\big(\,\mathcal{P}[\Delta_{\mathcal{C}}(\langle s,q\rangle, z)]\big) < 1$. But then $\Delta_{\mathcal{C}}(\langle s,q\rangle, z)$ cannot be a pure cut. Contradiction.

"$\Longleftarrow$": Suppose now that $(\langle s,q\rangle, z)$ has no extension. To prove that $C = \Delta_{\mathcal{C}}(\langle s,q\rangle, z)$ is a cut we rely on the second part of Lemma 4.45. Thus, the task is to show that $\Delta_{\mathcal{C}}(C,x) \neq \varnothing$ for all finite words $x \in S^+$ where x is a finite path in $\mathcal{M}$ starting in s. To prove this, one first shows that C is contained in some pure cut R of the form $\Delta_{\mathcal{C}}(\langle s,r\rangle, z)$ for some state $r \in Q$. As the last symbol of z is s, the elements of R have the form $\langle s,r\rangle$ for some $r \in Q$. For each state $p \in Q \setminus \{q\}$ where $\langle s,p\rangle \notin R$, there is some finite word y with

$$\langle s,q\rangle \xrightarrow{y}_{\mathcal{C}} \langle s,q\rangle \xrightarrow{y}_{\mathcal{C}} \langle s,p\rangle$$

For this we can rely on an adaption of Lemma 4.29. Since there exists no extension for $(\langle s,q\rangle, z)$, we have $\Delta_{\mathcal{C}}(\langle s,p\rangle, z) = \varnothing$ for all states $p \in Q$ where $\langle s,p\rangle \notin C$. This yields:

$$C \;=\; \Delta_{\mathcal{C}}(\langle s,q\rangle, z) \;=\; \Delta_{\mathcal{C}}(C,z) \;=\; \Delta_{\mathcal{C}}(R,z)$$

Using that R is a pure cut, we can now apply Lemma 4.45 to obtain the claim. □

To compute a pure cut in $\mathcal{C}$, we pick an arbitrary state $\langle s,q\rangle$ in $\mathcal{C}$ and successively generate path fragments $z_0, z_1, \ldots, z_k \in S^*$ in $\mathcal{M}$ by adding prefixes. More precisely, $z_0 = \varepsilon$ and z_{i+1} has the form yz_i for some $y \in S^+$ such that (1) $s\,y$ is a cycle in $\mathcal{M}$ and (2) there exists a state $p \in Q \setminus \{q\}$ in $\mathcal{U}$ with $\Delta_{\mathcal{C}}(\langle s,p\rangle, z_i) \neq \varnothing$ and $\{\langle s,q\rangle, \langle s,p\rangle\} \subseteq \Delta_{\mathcal{C}}(\langle s,q\rangle, y)$. Each such word y is called an extension of $(\langle s,q\rangle, z_i)$, and $\Delta_{\mathcal{C}}(\langle s,q\rangle, z_{i+1}) = \Delta_{\mathcal{C}}(\langle s,q\rangle, yz_i)$ is a proper superset of $\Delta_{\mathcal{C}}(\langle s,q\rangle, z_i)$. The set $C = \Delta_{\mathcal{C}}(\langle s,q\rangle, z)$ is a pure cut if and only if $(\langle s,q\rangle, z_i)$ has no extension. The search for an extension can be realized efficiently using a technique similar to the one presented in Section 4.1.3. Thus, after at most $\min\{|\mathcal{C}|, |Q|\}$ iterations, we obtain a pure cut C.

Having computed a pure cut C of $\mathcal{C}$, the values $\Pr_s^{\mathcal{M}}(\mathcal{P}[s,q])$ for $\langle s,q\rangle \in \mathcal{C}$ are then computable as the unique solution of the linear equation system consisting of equations (*) and the additional equation $\sum_{\langle s,q\rangle \in C} \zeta_{s,q} = 1$.

Theorem 4.50 (cf. Theorem 4.35)**.** *Let $\mathcal{C}$ be a positive BSCC of $\mathcal{P}$ and C a pure cut for $\mathcal{C}$. Then, the probability vector $(\Pr_{\mathcal{M}[s]}(\mathcal{L}_\omega(\mathcal{P}[s,q]))_{\langle s,q\rangle \in \mathcal{C}}$ is the unique solution of the following linear equation system:*

$$(1) \qquad \zeta_{s,q} \;=\; \sum_{t \in \mathrm{Post}(s)} \; \sum_{p \in \delta_{\mathcal{C}}(q,t)} P(s,t) \cdot \zeta_{t,p} \quad \textit{for all } \langle s,q\rangle \in \mathcal{C}$$

$$(2) \qquad \sum_{\langle s,q\rangle \in C} \zeta_{s,q} \;=\; 1$$

Proof. It is easy to see that the vector $(\zeta^*_{\langle s,q\rangle})_{\langle s,q\rangle \in \mathcal{C}}$ with

$$\zeta^*_{\langle s,q\rangle} \;=\; \Pr_{\mathcal{M}[s]}(\,\mathcal{L}_\omega(\mathcal{P}[s,q])\,)$$

is indeed a solution of (1) and (2). For the uniqueness, we rely on the proof presented for Theorem 4.35 in Section 4.1.3 with the $|\mathcal{C}| \times |\mathcal{C}|$-matrix M defined by $M_{\langle s,q\rangle,\langle t,p\rangle} = P(s,t)$ if $p \in \delta(q,t)$ and $M_{\langle s,q\rangle,\langle t,p\rangle} = 0$ otherwise.

□

Given a non-trivial, non-bottom SCC of $\mathcal{P}$ and a state $\langle s,q\rangle \in \mathcal{C}$, we write $\mathcal{C}[s,q]$ to denote the sub-UBA of $\mathcal{P}$ that arises when declaring $\langle s,q\rangle$ as initial state and restricting the transitions of $\mathcal{P}$ to those inside $\mathcal{C}$. That is, $\mathcal{C}[t,p] = \big(\mathcal{C}, S, \Delta_{\mathcal{C}}, \langle s,q\rangle, \mathrm{Inf}\,(F \cap \mathcal{C})\big)$. Then, $\mathcal{C}$ is positive iff $\Pr_{\mathcal{M}[s]}(\mathcal{C}[s,q]) > 0$ for some state $\langle s,q\rangle \in \mathcal{C}$ iff $\Pr_{\mathcal{M}[s]}(\mathcal{C}[s,q]) > 0$ for all states $\langle s,q\rangle \in \mathcal{C}$.

In this way we adapt Theorem 4.35 to obtain the values $\Pr_{\mathcal{M}[s]}(\mathcal{P}[s,q])$ for the states $\langle s,q\rangle$ belonging to some positive BSCC of $\mathcal{P}$. It remains to explain how to adapt the equation system of Theorem 4.40.

Recall that we assume a preprocessing that treats the SCCs of $\mathcal{P}$ in a bottom-up manner and turns $\mathcal{P}$ into a UBA where all BSCCs are non-trivial and positive. Q_{BSCC} denotes the set of states that are contained in some BSCC of $\mathcal{P}$ and $Q_?$ denotes the states of (the modified UBA) $\mathcal{P}$ not contained in Q_{BSCC}. For $\langle s,q\rangle \in Q_?$, let $\beta_{s,q} = 0$ if $\Delta(\langle s,q\rangle, t) \cap Q_{BSCC} = \varnothing$ for all $t \in S$. Otherwise:

$$\beta_{s,q} \;=\; \sum_{t \in \mathrm{Post}(s)} \; \sum_{\substack{p \,\in\, \delta(q,\,t) \text{ s.t.} \\ \langle t,p\rangle \,\in\, Q_{BSCC}}} P(s,t) \cdot \Pr_{\mathcal{M}[t]}(\,\mathcal{P}[t,p]\,)$$

Theorem 4.51 (cf. Theorem 4.40)**.** *Notations and assumptions as before. Then, the vector* $(\Pr_{\mathcal{M}[s]}(\mathcal{P}[s,q]))_{\langle s,q\rangle \in Q_?}$ *is the unique solution of the following linear equation system:*

$$\zeta_{s,q} \;=\; \sum_{t \in \mathrm{Post}(s)} \; \sum_{\substack{p \,\in\, \delta(q,\,t) \;\, s.t. \\ \langle t,p\rangle \,\notin\, Q_{BSCC}}} P(s,t) \cdot \zeta_{t,p} \;+\; \beta_{s,q} \qquad \textit{for } \langle s,q\rangle \in Q_?$$

Proof. It is easy to see that the vector $(\Pr_{\mathcal{M}[s]}(\mathcal{P}[s,q]))_{\langle s,q\rangle \in Q_?}$ indeed solves the linear equation system. For the uniqueness of the solution, we can apply the same arguments as in the proof of Theorem 4.40, but now for the $|Q_?| \times |Q_?|$-matrix M given by $M_{\langle s,q\rangle,\langle t,p\rangle} = P(s,t)$ if $p \in \delta(q,t)$ and $M_{\langle s,q\rangle,\langle t,p\rangle} = 0$ otherwise, where $\langle s,q\rangle$ and $\langle t,p\rangle$ range over all states in $Q_?$.

□

Let Q_{BSCC} be the set of BSCC states of $\mathcal{P}$ and $Q_?$ be the states of $\mathcal{P}$ not contained in Q_{BSCC}. For $\langle s,q\rangle \in Q_?$, let $\beta_{s,q} = 0$ if $\Delta(\langle s,q\rangle, t) \cap Q_{BSCC} = \varnothing$ for all $t \in S$. Otherwise:

$$\beta_{s,q} \;=\; \sum_{t \in \mathrm{Post}(s)} \; \sum_{\substack{p \,\in\, \delta(q,\,t) \text{ s.t.} \\ \langle t,p\rangle \,\in\, Q_{BSCC}}} P(s,t) \cdot \Pr_{\mathcal{M}[t]}(\,\mathcal{P}[t,p]\,)$$

Then, the vector $(\Pr_{\mathcal{M}[s]}(\mathcal{P}[s,q]))_{\langle s,q\rangle \in Q_?}$ is the unique solution of the linear equation system

$$\zeta_{s,q} \quad = \quad \sum_{t \in \mathrm{Post}(s)} \quad \sum_{\substack{p \,\in\, \delta(q,t) \text{ s.t.} \\ \langle t,p\rangle \,\notin\, Q_{BSCC}}} P(s,t) \cdot \zeta_{t,p} \quad + \quad \beta_{s,q} \qquad \text{for } \langle s,q\rangle \in Q_?$$

This completes the proof of Theorem 4.1.

4.1.6 Separated Büchi automata

Recall that an NBA is called separated if the languages of its states are pairwise disjoint. Obviously, each separated NBA is unambiguous. Although separated Büchi automata are as powerful as the full class of NBA [CM03] and translations of LTL formulas into separated UBA of (single) exponential time complexity exist [WVS83; CSS03], non-separated UBA and even deterministic automata can be exponentially more succinct than separated UBA [BL10].

We now explain in which sense our algorithm for (possibly non-separated) UBA can be seen as a conservative extension of the approach presented by Couvreur, Saheb and Sutre [CSS03]. To keep the presentation simple, we consider the techniques to compute $\Pr(\mathcal{L}_\omega(\mathcal{U}))$. Analogous statements hold for the computation of $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{U}))$ for a given Markov chain $\mathcal{M}$.

Given a separated, strongly connected UBA $\mathcal{U} = \big(Q, \Sigma, \delta, Q_0, \mathrm{Inf}\,(F)\big)$ with at least one initial and one final state, we have:

$$\Pr(\mathcal{L}_\omega(\mathcal{U})) > 0 \quad \text{iff} \quad Q \text{ is a reachable cut}$$

Furthermore, Q is a cut iff $\delta(Q,a) = Q$ for all $a \in \Sigma$ (Lemma 4.4). Thus, for the special case where the given UBA is separated and positive, there is no need for the inductive construction of a cut as outlined in Section 4.1.3. Instead, we can deal with $C = Q$. The linear equations in Theorem 4.35 can be derived from the results presented in [CSS03]. More precisely, equation (1) corresponds to the equation system in Proposition 5.1 of [CSS03], while equation (2) can be rephrased to $\sum_{q\in Q} \zeta_q = 1$, which corresponds to the equation used in Proposition 5.2 of [CSS03].

To check whether Q is a cut for a given (possibly non-positive) separated, strongly connected UBA, [CSS03] presents a simple criterion that is based on a counting argument. Lemma 4.14 in [CSS03] yields that for separated, strongly connected UBA we have:

$$Q \text{ is a cut} \quad \text{iff} \quad |\Sigma| \cdot |Q| \;=\; |\delta|$$

where $|\delta|$ is the total number of transitions in $\mathcal{U}$ given by $\sum\limits_{q\in Q} \sum\limits_{\sigma\in\Sigma} |\delta(q,\sigma)|$.

4.2 Implementation and Experiments

We implemented a probabilistic model checking procedure for Markov chains and UBA specifications using the algorithm detailed in Section 4.1 as an extension to the probabilistic model checker `PRISM` [KNP11] version 4.4 beta. All experiments were carried out on a computer with two Intel E5-2680 8-core CPUs at 2.70 GHz with 384GB of RAM running Linux, a time limit of 30 minutes and a memory limit of 10GB, if not stated otherwise. Our implementation is based on the `explicit` engine of `PRISM`, where the Markov chain is represented explicitly. Our implementation supports UBA-based model checking for handling the LTL fragment of `PRISM`'s PCTL*-like specification language as well as direct verification against a path specification given by a UBA provided in the HOA format [Bab+15]. For LTL formulas, we rely on external LTL-to-UBA translators. For the purpose of the benchmarks we employ the `ltl2tgba` tool from `SPOT` [Dur14] version 2.5 to generate a UBA for a given LTL formula. The according implementation, as well as the data logs can be found at [Mül18].

For the linear algebra parts of the algorithms, we rely on the `COLT` library [Hos04]. We considered two different variants for the SCC computations. The first variant (see page 73) relies on `COLT` to perform a QR decomposition of the matrix for the SCC to compute its rank, which allows to decide the positivity of the SCC. The second approach (see page 81) relies on a variant of the power iteration method for iteratively computing an eigenvector. This method has the benefit that, in addition to deciding the positivity, the computed eigenvector can be directly used to compute the values for a positive SCC, once a cut has been found. (As the proof of Theorem 4.35 shows: $\Pr(\mathcal{L}_\omega(q)) = \zeta^*_q / \sum_{p \in C} \zeta^*_p$ if ζ^* is an eigenvector of the matrix M for eigenvalue 1.) We evaluated the performance and scalability of the cut generation algorithm together with both approaches for treating SCCs. For this purpose we selected automata specifications that are challenging for our UBA-based model checking approach. As the power iteration method performed better, our benchmark results presented in this section are based on power iteration method for the SCC handling.

4.2.1 Evaluation of the rank computation

To assess the scalability of our implementation in the face of particularly difficult UBA, we considered two families of parametrized UBA. Both have an alphabet defined over a single atomic proposition resulting in a two-element alphabet that we use to represent either a 0 or a 1 bit. The first automaton ("complete automaton"), depicted in Figure 4.10 on the left for $k = 2$, is a complete automaton, i.e., recognizes Σ^ω. It consists of a single, accepting starting state that non-deterministically branches to one of 2^k states, each one leading after a further step to a k-state chain that only lets a particular k-bit bitstring pass, subsequently returning to the initial state. As all the k-bit bitstrings that can occur have a chain, the automaton is complete. Likewise, the automaton is unambiguous as each of the bitstrings can only pass via one of the chains.

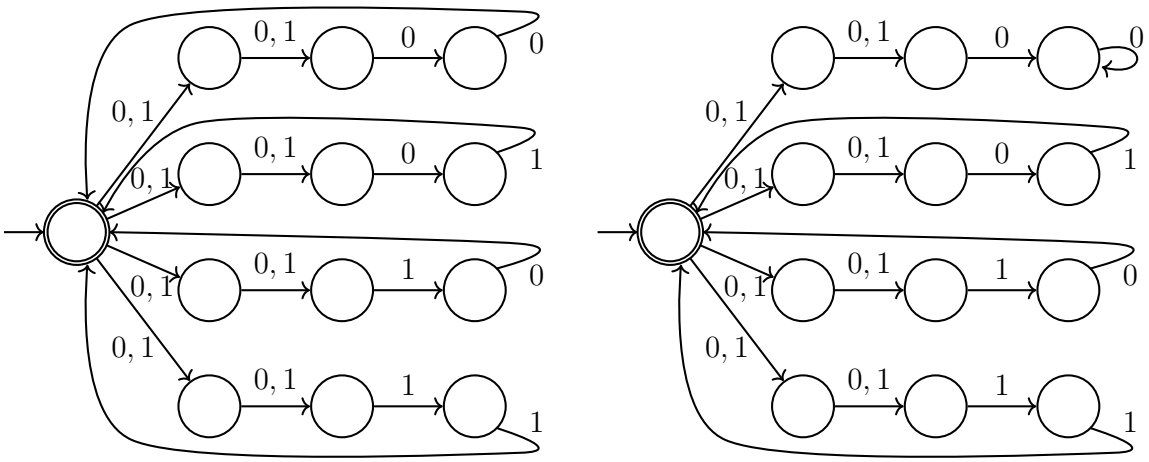


Figure 4.10: UBA "complete automaton" (left) and "nearly complete automaton" (right) for $k = 2$.

Our second automaton ("nearly complete automaton"), depicted in Figure 4.10 on the right for $k = 2$, arises from the first automaton by a modification of the chain for the "all zero" bitstring, inhibiting the return to the initial state. Clearly, the automaton is not complete.

We use both kinds of automata in an experiment using our extension of `PRISM` against a simple, two-state DTMC that encodes a uniform distribution between the two "bits". This allows us to determine whether the given automaton is almost universal. As the `PRISM` implementation requires the explicit specification of a DTMC, we end up with a product that is slightly larger than the UBA, even though we are essentially performing the UBA computations for the uniform probability distribution. In particular, this experiment serves to investigate the scalability of our implementation in practice for determining whether an SCC is positive, for the cut generation and for computing the probabilities for the SCC states. It should be noted that equivalent deterministic automata, e.g., obtained by determinizing the UBA using the `ltl2dstar` tool are significantly smaller (in the range of tens of states) due to the fact that the UBA in question are constructed inefficiently on purpose.

			cut generation			power iter.		rank-based	
k	$\lvert\mathcal{A}\rvert$	SCC size	t_{cut}	ext. checks	cut size	t_{eigen}	iter.	t_{positive}	t_{values}
5	193	258	0.1 s	10124	32	$<$0.1 s	215	0.5 s	0.4 s
6	449	578	0.1 s	40717	64	$<$0.1 s	282	4.3 s	4.3 s
7	1025	1282	0.9 s	172102	128	0.1 s	358	56.5 s	56.9 s
8	2305	2818	1.8 s	929413	256	0.1 s	443	830.8 s	835.1 s
9	5121	6146	17.9 s	6818124	512	0.1 s	537	-	-

Table 4.1: Benchmark results for "complete automaton" with parameter k. − stands for timeout.

Table 4.1 presents statistics for our experiments with the "complete automaton"

			power iteration		rank-based
k	$\lvert\mathcal{A}\rvert$	SCC size	t_{eigen}	iter.	t_{positive}
5	193	250	<0.1 s	52	0.4 s
6	449	569	<0.1 s	78	4.1 s
7	1025	1272	<0.1 s	112	54.4 s
8	2305	2807	0.1 s	155	844.0 s
9	5121	6134	0.1 s	205	-

Table 4.2: Benchmark results for "nearly complete automaton" with parameter k.

with various parameter values k, resulting in increasing sizes of the UBA and the SCC (number of states). We list the time spent for generating a cut (t_{cut}), the number of checks whether a given word is an extension during the cut generation algorithm, and the size of the cut. In all cases, the cut generation requires 2 iterations. Then we compare the SCC handling based on the power iteration with the SCC handling relying on a rank computation for determining positivity of the SCC and a subsequent computation of the values. For the power iteration method, we provide the time spent for iteratively computing an eigenvector (t_{eigen}) and the number of iterations (iters.). For the other method, we provide the time spent for the positivity check by a rank computation with a QR decomposition from the `COLT` library (t_{positive}) and for the subsequent computation of the values via solving the linear equation system (t_{values}). We used an overall timeout of 60 minutes for each `PRISM` invocation and an epsilon value of 10^{-10} as the convergence threshold.

As can be seen, the power iteration method for the numeric SCC handling performed well, with a modest increase in the number of iterations for rising k until converging on an eigenvector, as it can fully exploit the sparseness of the matrix. In contrast, the QR decomposition for rank computation performs worse. The time for cut generation exhibits a super-linear relation with k, which is reflected in the higher number of words that were checked to determine that they are an extension. Note that our example was chosen in particular to put stress on the cut generation.

The results for the "nearly complete automaton", depicted in Table 4.2, focus on the computation in the "dominant SCC", i.e., the one containing all the chains that return to the initial state. For the other SCC, containing the self-loop, non-positivity is immediately clear as it does not contain a final state. In contrast to the "complete automaton", no cut generation takes place, as the SCC is not positive. The results roughly mirror the ones for the "complete automata", i.e., the power iteration method is quite efficient in determining that the SCC is not positive, while the QR decomposition for the rank computation needs significantly more time and scales worse.

4.2.2 Case Study: Bounded Retransmission Protocol

We report here on benchmarks using the bounded retransmission protocol (BRP) case study of the `PRISM` benchmark suite [KNP12]. The model from the benchmark

suite covers a single message transmission, retrying for a bounded number of times in case of an error. We have slightly modified the model to allow the transmission of an infinite number of messages by restarting the protocol once a message has been successfully delivered or the bound for retransmissions has been reached. We will include benchmarks with pre-generated automata, as well as benchmarks with LTL as starting point. For the LTL benchmarks we have implemented the iterative refinement method presented in [CY95], as it is also a single-exponential approach. We call this implementation `PRISM` CY95. The benchmarks include also an evaluation for deterministic Emerson-Lei automata generated by `Delag`, which we present in Chapter 5.

Automata based specifications. We consider the property "the message was retransmitted k steps before an acknowledgment." To remove the effect of selecting specific tools for the LTL-to-automaton translation (`ltl2tgba` for UBA, the Java-based `PRISM` reimplementation of `ltl2dstar` [KB06] to obtain a deterministic Rabin automaton (DRA) for the standard `PRISM` approach), we consider directly model checking against automata specifications at first. As the language of the property is equivalent to the UBA depicted in Figure 4.1 (on the left) when a stands for a retransmission, b for an acknowledgment, and c for no acknowledgment, we use this automaton and the minimal DBA for the language (this case is denoted by $\mathcal{A}^k$). We additionally consider the UBA and DBA obtained by replacing the self-loop in the last state with a switch back to the initial state (denoted by $\mathcal{B}^k$), i.e., roughly applying the ω-operator to $\mathcal{A}^k$.

Table 4.3 shows results for selected k (with a timeout of 30 minutes), demonstrating that for this case study and properties our UBA-based implementation is generally competitive with the standard approach of `PRISM` relying on deterministic automata. For $\mathcal{A}^k$, our implementation detects that the UBA has a special shape where all final states have a true-self loop which allows skipping the SCC handling. Without this optimization, t_{Pos} are in the sub-second range for all considered $\mathcal{A}^k$. At a certain point, the implementation of the standard approach in `PRISM` becomes unsuccessful, due to `PRISM` size limitations in the product construction of the Markov chain and the deterministic automaton ($\mathcal{A}^k/\mathcal{B}^k$: $k \geqslant 16$). As can be seen, using the UBA approach we were able to successfully scale the parameter k beyond 48 when dealing directly with the automata-based specifications ($\mathcal{A}^k/\mathcal{B}^k$) and within reasonable time required for model checking.

LTL based specifications. We consider here two LTL properties: The first one is:

$$\varphi^k = (\neg\texttt{ack\_received})\ \mathcal{U}\ \big(\texttt{retransmit} \wedge (\neg\texttt{ack\_received}\ \mathcal{U}^{=k}\ \texttt{ack\_received})\big),$$

where $a\,\mathcal{U}^{=k}b$ stands for $a \wedge \neg b \wedge \bigcirc(a \wedge \neg b) \wedge \ldots \wedge \bigcirc^{k-1}(a \wedge \neg b) \wedge \bigcirc^k b$. The formula φ^k ensures that k steps before an acknowledgment the message was retransmitted. Hence, it is equivalent to the property described by the automaton $\mathcal{A}^k$. For the LTL-to-automata translation we included the Java-based `PRISM` reimplementation

	PRISM standard			PRISM UBA			
	$\lvert\mathcal{A}^k_{DRA}\rvert$	$\lvert\mathcal{M}\otimes\mathcal{A}^k_{DRA}\rvert$	t_{MC}	$\lvert\mathcal{A}^k_{UBA}\rvert$	$\lvert\mathcal{M}\otimes\mathcal{A}^k_{UBA}\rvert$	t_{MC}	t_{Pos}
$k=4$, $\mathcal{A}^4$	33	61,025	0.4 s	6	34,118	0.3 s	
$\mathcal{B}^4$	33	75,026	0.4 s	6	68,474	1.3 s	1.0 s
$k=6$, $\mathcal{A}^6$	129	62,428	0.5 s	8	36,164	0.2 s	
$\mathcal{B}^6$	129	97,754	0.5 s	8	99,460	1.7 s	1.3 s
$k=8$, $\mathcal{A}^8$	513	64,715	0.6 s	10	38,207	0.3 s	
$\mathcal{B}^8$	513	134,943	0.7 s	10	136,427	2.6 s	2.1 s
$k=14$, $\mathcal{A}^{14}$	32,769	83,845	4.2 s	16	44,340	0.3 s	
$\mathcal{B}^{14}$	32,769	444,653	4.9 s	16	246,346	6.8 s	6.1 s
$k=16$, $\mathcal{A}^{16}$	131,073	–	–	18	46,390	0.3 s	
$\mathcal{B}^{16}$	131,037	–	–	18	282,699	8.9 s	8.0 s
$k=48$, $\mathcal{A}^{48}$	–	–	–	50	79,206	0.8 s	
$\mathcal{B}^{48}$	–	–	–	50	843,414	72.4 s	70.3 s

Table 4.3: Statistics for DBA/DRA- and UBA-based model checking of the BRP case study (parameters $N = 16$, $MAX = 128$), a DTMC with 29358 states, depicting the number of states for the automata and the product and the time for model checking (t_{MC}). For $\mathcal{B}$, the time for checking positivity (t_{Pos}) is included in t_{MC}. The mark − stands for "not available" or timeout (30 minutes).

of `ltl2dstar` [KB06] to obtain a deterministic Rabin automaton (DRA) for the standard `PRISM` approach as well as our tool `Delag` which we present in Chapter 5. For the purpose of this benchmark, it is sufficient to know that `Delag` recognizes that φ^k is a cosafety language and creates a deterministic Büchi automaton with a construction from `Rabinizer` [EKS16]. For the generation of UBA, we relied on `SPOT`, as it is the only tool that is capable of generating UBA explicitly. `SPOT` actually generates a UFA for φ^k which is recognized by our implementation in `PRISM` as explained above.

k	PRISM standard			PRISM Delag			PRISM UBA			PRISM CY95	
	$\lvert\mathcal{A}_{DRA}\rvert$	$\lvert\mathcal{M}\otimes\mathcal{A}_{DRA}\rvert$	t_{MC}	$\lvert\mathcal{A}_{EL}\rvert$	$\lvert\mathcal{M}\otimes\mathcal{A}_{EL}\rvert$	t_{MC}	$\lvert\mathcal{A}_{UBA}\rvert$	$\lvert\mathcal{M}\otimes\mathcal{A}_{UBA}\rvert$	t_{MC}	$\lvert\mathcal{M}_{final}\rvert$	t_{MC}
4	122	29,358	0.6 s	23	29,358	1.1 s	21	34,757	0.3 s	42,081	1.8 s
6	4,602	29,358	1.8 s	73	61,790	1.2 s	71	37,951	0.4 s	61,795	3.8 s
8	–	–	–	267	63,698	1.3 s	265	41,902	1.1 s	87,063	9.2 s
14	–	–	–	16,401	79,576	6.4 s	–	–	–	106,876	28.1 s
16	–	–	–	65,555	–	–	–	–	–	106,876	35.1 s

Table 4.4: Statistics for automata-based (standard, Delag, and UBA) and CY95 model checking of the BRP model and φ^k. For every approach except PRISM CY95, the corresponding automata sizes and product sizes are depicted, for PRISM CY95 the size of the last refined Markov chain ($\lvert\mathcal{M}_{final}\rvert$) is depicted. For every approach the overall model checking times (t_{MC}) are listed, which includes the time for automata translation in case of the automata based approaches.

k	PRISM standard			PRISM Delag			PRISM UBA					PRISM CY95	
	$\lvert\mathcal{A}_{DRA}\rvert$	$\lvert\mathcal{M}\otimes\mathcal{A}_{DRA}\rvert$	t_{MC}	$\lvert\mathcal{A}_{EL}\rvert$	$\lvert\mathcal{M}\otimes\mathcal{A}_{EL}\rvert$	t_{MC}	$\lvert\mathcal{A}_{UBA}\rvert$	$\lvert\mathcal{M}\otimes\mathcal{A}_{UBA}\rvert$	t_{pos}	t_{Cut}	t_{MC}	$\lvert\mathcal{M}_{final}\rvert$	t_{MC}
1	6	29,358	0.3 s	9	33,454	1.0 s	4	31,422	$<$0.1 s	n/a	0.3 s	29,358	0.5 s
2	17	37,678	0.4 s	16	39,726	1.1 s	8	41,822	4.5 s	0.2 s	5.0 s	33,470	0.5 s
3	65	39,726	0.4 s	28	43,806	1.1 s	14	45,934	4.9 s	0.2 s	5.5 s	37,711	0.8 s
4	314	43,806	0.5 s	52	47,902	1.2 s	22	54,126	5.6 s	0.2 s	6.2 s	42,081	1.1 s
5	1,443	47,902	1.1 s	100	56,062	1.3 s	32	62,334	6.4 s	0.2 s	7.0 s	50,918	1.3 s
6	9,016	56,029	3.9 s	196	60,113	1.3 s	44	78,669	9.1 s	0.3 s	9.9 s	61,795	1.7 s
7	67,964	–	–	388	68,249	1.5 s	58	86,853	10.1 s	0.4 s	11.0 s	74,952	2.2 s
8	–	–	–	772	72,323	1.9 s	74	103,157	13.4 s	0.2 s	14.4 s	88,117	2.7 s
10	–	–	–	3,076	84,468	5.7 s	112	127,562	19.2 s	0.2 s	21.2 s	88,117	3.7 s

Table 4.5: Statistics for automata-based (standard, Delag, and UBA) and CY95 model checking of the BRP model and ψ^k. The structure of this table corresponds to Table 4.4, but with additional listing of the time for the positivity checks t_{pos} and cut calculation time t_{cut}. n/a means not available.

Table 4.4 lists the results for model checking φ^k. From a certain point on, the implementation of the standard approach in PRISM is unsuccessful, due to PRISM restriction in the DRA construction ($k \geqslant 8$) or a timeout during UBA construction ($k \geqslant 13$). If PRISM (vs. SPOT or Delag) was able to construct an automaton, then the automata-based approaches were faster than PRISM CY95. Delag comes close to produce deterministic automata with the possible 2^k states, but SPOT produces unnecessary large UBA for φ^k. This is reflected in the model checking runtimes where the UBA approach is the fastest for $k \leq 8$, but this turns for $k \geq 9$, where the UBA approach took $3.1s$, and PRISM Delag took $1.3s$. For $k \geq 13$, SPOT was not able to produce a UBA within the time bound of 30 min.

As second formula we investigate

$$\psi^k = \Box(\texttt{msg\_send} \rightarrow \Diamond(\texttt{ack\_send} \wedge \Diamond^{\leqslant k}\texttt{ack\_received})),$$

where $\Diamond^{\leqslant k}a$ denotes $a \vee \underbrace{\bigcirc(a \vee \bigcirc(\ldots \vee \bigcirc a))}_{k \text{ times}}$. This formula requires that every request (sending a message and waiting for an acknowledgment) is eventually responded by an answer (the receiver of the message sends an acknowledgment and this acknowledgment is received within the next k steps).

Table 4.5 summarizes the result of the benchmark for ψ^k. Here, the PRISM standard approach with its own implementation of ltl2dstar was able to finish the calculations until $k = 6$. For $k = 7$, PRISM standard was able to construct the DRA (with 67,964 states and within 27.5 seconds), but not able to construct the product anymore. In case of Delag, ψ^k does not belong to the natively supported fragment, and SPOT is used as the fallback solution to generate a deterministic automaton.

In contrast to the deterministic automata, the UBA sizes increase moderately for an increasing k. The positivity check of the UBA-approach is the most time-consuming part of the calculation. For $k = 1$ there is no positive SCC, so the cut calculation is omitted. The model checking process consumes more time in the UBA case in comparison with PRISM standard until $k = 6$, but for bigger k the performance turned around. Even if PRISM standard would have complete the calculation for $k = 7$ it would have been slower, as the creation of the DRA took 27.5 seconds.

The other single exponential approach, PRISM CY95, outperforms the other approaches in case of ψ^k. The final refined Markov chain $\mathcal{M}_{final}$ are always smaller than the size of the product with the UBA. So, besides the case $k = 1$, the ratio between t_{MC} for PRISM UBA and t_{MC} for PRISM CY95 amounts between 5 ($k = 7$) and 10 ($k = 2$).

4.2.3 NBA versus UBA

To gain some understanding on the cost of requiring unambiguity for an NBA, we have compared the sizes of NBA and UBA generated by the ltl2tgba tool of SPOT for the formulas of [EH00; SB00; DAC99] used for benchmarking, e.g., in [KB06]. We consider both the "normal" formulas and their negations, yielding 188 formulas.

Number of states $\leq x$	≤ 1	≤ 2	≤ 3	≤ 4	≤ 5	≤ 7	≤ 10	≤ 12	≤ 20	> 20
`ltl2tgba` NBA	12	49	103	145	158	176	181	185	188	0
`ltl2tgba` UBA	12	42	74	108	123	153	168	173	180	8

Table 4.6: Number of formulas where the (standard) NBA and UBA has a number of states $\leq x$.

As can be seen in Table 4.6, both the NBA and UBA tend to be of quite reasonable size. Most of the generated UBA (102) have the same size as the NBA and for 166 of the formulas the UBA is at most twice the size as the corresponding NBA. The largest UBA has 112 states, the second largest has 45 states.

4.3 Conclusion

In this chapter we presented a polynomial-time algorithm for Markov chain analysis against properties given as unambiguous Büchi automata. As LTL formulas can be transformed into UBA with a single exponential blow-up, the overall time complexity becomes single exponential.

We provided an extension of `PRISM` for this approach and its practical evaluation shows that the UBA-based approach is very competitive in comparison to the approach using deterministic automata, in some cases even outperforms deterministic automata.

For the other single exponential approaches like [CSS03] using separated automata and [BRV04] using weak alternating automata we are not aware of any available implementation.[5]

The single exponential approach of [CY95] is competitive to the UBA approach, as specifically the positivity check can be eluded. In our benchmark of the bounded retransmission protocol, the UFA/UBA approach outperformed `PRISM` CY95, if the property was actually a UFA, or the LTL formula could be translated to one. In case an actual UBA was used, `PRISM` CY95 outperformed `PRISM` UBA due to the long positivity checks to identify the positive SCCs.

In a more abstract comparison, Markov chain analysis via UBA offer some advantages regarding the flexibility. The intermediate step of transforming LTL to UBA allows reduction techniques as, for example, simulation. Also, the generation of UBA could be designed to create smaller UBA. As our experiments suggests, the eigenvalue algorithm can deduce non-positiveness of an SCC very fast in practice, which is also reflected in the benchmark of the bounded retransmission protocol. In case of a positive SCC, the eigenvalue computation consumes more time and the UBA approach can be slower than the iterative refinement of `PRISM` CY95. For the generation of UBA we used the tool `SPOT`, which implements a rather simple and straightforward way to produce unambiguous Büchi automata [Dur17]. Alternatively,

[5] The paper [CSS03] addresses experiments with a prototype implementation, but this implementation seems not to be available anymore. Section 4.1.6 briefly explains that our algorithm can be seen as a generalization of the approach of [CSS03] with separated UBA.

`Tulip` contained an LTL-to-UBA translator, but it is not available anymore. As in case with deterministic automata, the performance of the UBA-based approach depends strongly on the availability of small UBA. In contrast to non-deterministic or deterministic automata, the generation of small UBA or their simplification has not yet been explored thoroughly. Another promising approach would be to modify known LTL-to-NBA translations. We see a potential in the compositional approach of [EKS18] by a modification of the translations for the fragments to generate UBA and a disjoint union for the languages in the master theorem of [EKS18]. Alternatively, one would change the longer known translation of [GO01; Bab+12] such that one creates an unambiguous very weak alternating automaton and applies an adjusted, unambiguity-preserving transformation to NBA. However, for general LTL benchmarks, the difference in the number of states between NBA and UBA is comparably small, but as our LTL formulas for the bounded retransmission protocol benchmark suggest, the look-ahead of UBA has not been fully exploited yet.

5 Emerson-Lei acceptance

In the previous chapters we focused on ω-automata with a restricted form of non-determinism. Now we turn to deterministic ω-automata but delve into the acceptance condition of ω-automata. It is well-known, that the acceptance condition of an ω-automaton has an important influence of the size of the automaton, even in the non-deterministic case (see for example [Bok17b]). Also for deterministic automata, an exponential blow-up in the state-space may be unavoidable, if one switches from a Rabin acceptance to a Streett acceptance (or the other way around), see [SV89].

We reexamine the Muller acceptance, but due to its large representation, we add a crucial twist by a symbolic representation. A (standard) Muller acceptance is a subset of the state power set (or transition power set) stating explicitly which $\inf(\rho)$ of a run ρ are accepting. More formally, a state-based Muller acceptance of an automaton with state space Q is a set $Z \subseteq 2^Q$ and a run ρ is accepting if and only if $\inf(\rho) \in Z$. Thus, the acceptance can be exponentially big in the number of states. This has led to a representation in a symbolic fashion, as presented in [Bab+15], which we call Emerson-Lei acceptance (EL for short). For the symbolic representation the atoms $\mathrm{Fin}(Z)$ and $\mathrm{Inf}(Z)$ with $Z \subseteq Q$ are introduced with the obvious semantics: A run ρ is accepting for $\mathrm{Fin}(Z)$ if and only if Z is visited only finitely often in ρ; ρ is accepting for $\mathrm{Inf}(Z)$ if and only if Z is visited infinitely often in ρ.[1]

This symbolic Muller acceptance have already been studied by Emerson and Lei. In [EL85] they regard it as a fairness constraint for CTL model checking on Kripke structures. One problem they consider is to search for a path satisfying a fairness constraint (which corresponds to checking non-emptiness for an EL automaton and to positive probability under a maximal scheduler for an MDP). In case of disjunction of Streett conditions they provide an algorithm with a linear time complexity in the Kripke structure size and quadratic in the size of the fairness constraint. The authors also reduces 3SAT to the general case of finding an accepting path in a Kripke structure for a fairness constraint and thus show the NP-hardness of it. The NP membership of the problem is obtained by a reference to [SC85]. The authors of [SC85] consider Kripke structures and the LTL fragment $\mathrm{LTL}(\Diamond, \Box)$, i.e., the fragment allowing only $\Diamond$ and $\Box$ as temporal operators and prove an analogous NP-completeness result for this fragment.

Moving to a compactly expressed acceptance condition allows us to reduce the number of states and to use fewer acceptance sets compared to existing translations, although there is a well-known exponential lower bound for the size of deterministic

[1] Transition-based Muller acceptance and transition-based Emerson-Lei acceptance are defined analogously.

ω-automata starting from a non-deterministic ω-automaton [Mic88; Saf88].

The richer acceptance condition complicates typical algorithmic questions, as deciding positivity for Markov decision processes (does "$\Pr^{\max}_{\mathcal{M}}(L) > 0$" hold), or emptiness for ω-automata.

While for Rabin acceptance the solution of the positivity problem for MDPs (and emptiness for ω-automata) is rather straightforward, the same problems for Streett acceptance triggered a long line of research, see, e.g., [EL85; EL87; BGC09b] where a repeated generation of SCCs and removal of $\mathrm{Fin}(\cdot)$ states is done, until one SCC is found, that contains only $\mathrm{Inf}(\cdot)$ states or no SCC is left, which means that the Streett condition is unsatisfiable. This algorithm has been improved by heuristics in [RT96; LH00], and adapted to symbolic computation in [BGS00], parallel computation [ČP03], and on-the-fly-techniques [DPC09].

A lot of research has put into the translation of LTL to deterministic ω-automata (see Section 1.1.3), but to the best of our knowledge all those translations are targeted at a particular acceptance like Rabin, Streett or parity. There exists already the idea of a production construction in [Bab+15] to obtain a complex acceptance: The formula is split into several subformulas, where the top-most operator in the syntax tree is a temporal operator, and the subformulas are not nested within the scope of another temporal operator. These subformulas are converted to deterministic automata with one of the already known translators. These deterministic automata are composed in a product construction, where the graph is standard product, and the acceptance of the product automaton where every top-most temporal formula is substituted by the acceptance of the corresponding deterministic automaton.

Contribution. We present a translation from LTL to deterministic Emerson-Lei automata that trades a compact state space for a more complex acceptance condition structure. Here, we give a direct translation of fragments of LTL without an intermediate step over non-deterministic automata. We consider special fairness properties in particular and give a translation based on buffers. For safety and cosafety LTL formulas we rely on the *af* function [EKS16; Sic+16] computing the left-derivative directly on the formula. Additionally, if we encounter a subformula not contained in our supported fragments for a direct translation, we rely on external tools for translation, and compose a deterministic automaton for the overall formula with the product construction of [Bab+15]. We improve the product construction and make use of the knowledge about the subformulas, if one subformula falls into the fairness or safety/cosafety fragment. This knowledge helps us to suspend one automaton similar to [Bab+13a], but we work more on the automaton level whereas [Bab+13a] works on the LTL level. A general scheme for our approach is depicted in Figure 5.1, which we implemented in our tool `Delag` (Deterministic Emerson-Lei Automata Generator).

As a second theoretical contribution we consider the complexity of deciding $\Pr^{\max}_{\mathcal{M}}(\Phi) > 0$ for an MDP $\mathcal{M}$ and an Emerson-Lei acceptance Φ. We prove, that the positivity problem is NP-complete for a class of Emerson-Lei acceptances if the satisfiability problem is NP-complete for the corresponding class of Boolean formu-

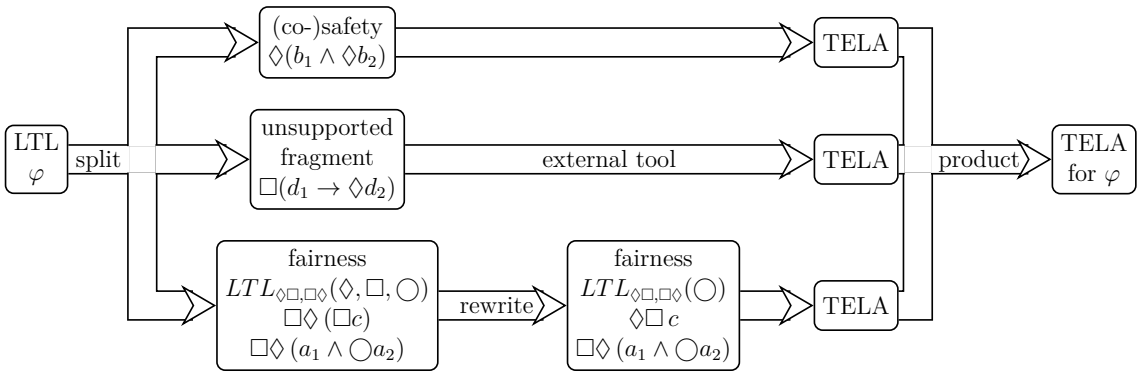


Figure 5.1: The input LTL formula is split up, each subformula is translated independently, and then a product automaton is constructed, as can be seen for the example $\varphi = \square\Diamond\,(a_1 \wedge \bigcirc a_2) \wedge \Diamond\,(b_1 \wedge \Diamond\, b_2) \wedge \square\Diamond\,(\square c) \wedge \square\,(d_1 \rightarrow \Diamond d_2)$. The abbreviation TELA stands for transition-based Emerson-Lei automata.

las. This NP-completeness result inspired us to take on a well-known SAT-solving algorithm, called DPLL (see [DP60] and its refinement [DLL62]). This DPLL-based algorithm for deciding positivity for the whole class of Emerson-Lei acceptance turns out to be polynomial in time for Streett and (generalized) Rabin acceptance.

We conducted several experiments to evaluate the practical impact of this idea: At first we compared the size of the automata measured in state space size as well as acceptance sizes for our tool and several other tools like `SPOT` and `Rabinizer`. Secondly, we performed a case study (IEEE 802.11 Wireless LAN Handshaking protocol) and also compared it with `SPOT` and `Rabinizer`. On both sides, we could show the potential of `Delag`, i.e., allowing arbitrary acceptance conditions to obtain smaller automata.

5.1 Construction

The automaton is constructed from an LTL formula in positive normal form as a product of smaller automata for each top-most temporal subformula. For this we define $\mathsf{sf}(\varphi)$ to be the set of temporal subformulas not nested within the scope of another temporal operator, e.g., $\mathsf{sf}((\Diamond\square a) \vee \bigcirc b) = \{\Diamond\square a, \bigcirc b\}$.

5.1.1 Fragments of LTL

We study several syntactic fragments of LTL. Recall, that LTL(M) consists all LTL formulas whose temporal operators occur solely in M, and that $\mathrm{LTL}_{\square\Diamond,\,\Diamond\square}(\Diamond, \square, \bigcirc)$ is an abbreviation for the set of formulas of the form $\Diamond\square\varphi$ or $\square\Diamond\varphi$ with φ having only $\Diamond, \square$ or $\bigcirc$ as temporal operators. At first, we show that the fairness LTL fragment can be simplified to formulas without nested $\Diamond$ and $\square$:

Theorem 5.1 (Fairness LTL Normal Form)**.** *Let φ be an $LTL_{\Diamond\Box,\Box\Diamond}(\Diamond, \Box, \bigcirc)$ formula. Then there exists an equivalent formula $\varphi' \equiv \varphi$ that is a Boolean combination of formulas in $LTL_{\Diamond\Box,\Box\Diamond}(\bigcirc)$ and in positive normal form.*

Proof. W.l.o.g. we assume that φ is given in positive normal form. Then, the exhaustive application of the following folklore equivalence-preserving rewrite rules, described in [EH00; SB00; Li+16b], brings every fairness LTL formula into the desired normal form:

$$
\begin{array}{rcl@{\qquad}rcl}
\Diamond\Box(\Diamond\varphi) & \mapsto & \Box\Diamond\varphi & \Box\Diamond(\Diamond\varphi) & \mapsto & \Box\Diamond\varphi \\
\Diamond\Box(\Box\varphi) & \mapsto & \Diamond\Box\varphi & \Box\Diamond(\Box\varphi) & \mapsto & \Diamond\Box\varphi \\
\Diamond\Box(\bigcirc\varphi) & \mapsto & \Diamond\Box\varphi & \Box\Diamond(\bigcirc\varphi) & \mapsto & \Box\Diamond\varphi \\
\Diamond\Box(\varphi \wedge \psi) & \mapsto & \Diamond\Box\varphi \wedge \Diamond\Box\psi & \Box\Diamond(\varphi \vee \psi) & \mapsto & \Box\Diamond\varphi \vee \Box\Diamond\psi \\
\Diamond\Box(\varphi \vee \Diamond\psi) & \mapsto & \Diamond\Box\varphi \vee \Box\Diamond\psi & \Box\Diamond(\varphi \wedge \Diamond\psi) & \mapsto & \Box\Diamond\varphi \wedge \Box\Diamond\psi \\
\Diamond\Box(\varphi \vee \Box\psi) & \mapsto & \Diamond\Box\varphi \vee \Diamond\Box\psi & \Box\Diamond(\varphi \wedge \Box\psi) & \mapsto & \Box\Diamond\varphi \wedge \Diamond\Box\psi \\
\varphi \notin \mathrm{LTL}(\bigcirc) \Rightarrow \Diamond\Box(\varphi) & \mapsto & \Diamond\Box(\mathsf{cnf}(\varphi)) & \varphi \notin \mathrm{LTL}(\bigcirc) \Rightarrow \Box\Diamond(\varphi) & \mapsto & \Box\Diamond(\mathsf{dnf}(\varphi))
\end{array}
$$

with $\mathsf{cnf}(\varphi)$ and $\mathsf{dnf}(\varphi)$ denoting the translation into conjunctive normal form (CNF) and disjunctive normal form (DNF). □

This translation might cause an exponential blow-up in formula size due to the translation into conjunctive and disjunctive normal form. Concerning the number of states, the construction for fairness LTL to deterministic automata we present is only dependent on the size of the alphabet and the nesting depth of the $\bigcirc$-operators, which are both unchanged (or even decreased) by the translation. However, as the formula after the translation can be exponentially big, the acceptance of the automaton can be exponentially big as well. Further, from now on we assume all fairness LTL formulas are rewritten to this normal form.

Apart from the rules listed above, our implementation uses several well-known simplification rules to rewrite formulas outside of the fairness fragment to formulas within, e.g., $\Box\Diamond(\varphi \,\mathcal{U}\, \psi) \mapsto \Box\Diamond\psi$ and $\Diamond\Box(\varphi \,\mathcal{U}\, \psi) \mapsto \Box\Diamond\psi \wedge \Diamond\Box(\varphi \vee \psi)$.

5.1.2 Fairness-LTL

We now show that there is a natural way to represent formulas of the LTL fairness fragment as deterministic automata. In particular, if we look at Boolean combinations of fairness-LTL formulas ($\mathrm{LTL}_{\Diamond\Box,\Box\Diamond}(\bigcirc)$), we obtain an acceptance condition mirroring the structure of the input formula. Furthermore, if the formula does not contain any $\bigcirc$, the automaton we obtain is a single-state automaton. For all other formulas we need to store a bounded history in the form of a FIFO-buffer of sets of atomic propositions (or valuations) that have been seen. We establish now the tools necessary to compute the structure of such a buffer. We use the following operations defined on finite and infinite sequences of sets (assuming $n \leq m$):

$$
\begin{aligned}
\text{Pointwise Intersection: } & \sigma_0\sigma_1 \ldots \sqcap \upsilon_0 \ldots \upsilon_m &&= (\sigma_0 \cap \upsilon_0) \ldots (\sigma_m \cap \upsilon_m)\varnothing^\omega \\
\text{Pointwise Union: } & \sigma_0 \ldots \sigma_n \sqcup \upsilon_0 \ldots \upsilon_m &&= (\sigma_0 \cup \upsilon_0) \ldots (\sigma_n \cup \upsilon_n) \ldots \upsilon_m \\
\text{Forward Closure: } & \mathsf{cl}(\sigma_0 \ldots \sigma_n) &&= \sigma_0(\sigma_0 \cup \sigma_1) \ldots \bigcup_{k=0}^{n} \sigma_k \\
\text{Drop Last Set of Letters: } & \mathsf{drop}(\sigma_0 \ldots \sigma_n\sigma_{n+1}) &&= \sigma_0 \ldots \sigma_n
\end{aligned}
$$

Relevant History. Let us consider an example formula: $\Box\Diamond(a_1 \wedge \bigcirc a_2)$. In order to check whether $w \models a_1 \wedge \bigcirc a_2$ holds we just need to know whether $a_1 \in w[0]$ and $a_2 \in w[1]$ holds. The rest of w can be ignored. The *relevant history* $\mathcal{H}(\varphi)$ for an LTL formula φ is a finite word over 2^{AP} and masks all propositions that are irrelevant for evaluating φ. We compute the relevant history $\mathcal{H}$ recursively from the structure of the formula:

$$
\mathcal{H}\colon \mathrm{LTL}(\bigcirc) \to (2^{AP})^* \qquad
\begin{array}{rclrcl}
\mathcal{H}(\mathit{true}) & = & \epsilon & \mathcal{H}(\mathit{false}) & = & \epsilon \\
\mathcal{H}(a) & = & \{a\} & \mathcal{H}(\neg a) & = & \{a\} \\
\mathcal{H}(\varphi \wedge \psi) & = & \mathcal{H}(\varphi) \sqcup \mathcal{H}(\psi) & \mathcal{H}(\varphi \vee \psi) & = & \mathcal{H}(\varphi) \sqcup \mathcal{H}(\psi)
\end{array}
\qquad \mathcal{H}(\bigcirc\varphi) = \varnothing\mathcal{H}(\varphi)
$$

Lemma 5.2. *Let φ be an LTL($\bigcirc$) formula and let $w = \sigma_0\,\sigma_1\,\ldots$ be a ω-word. Then $w \models \varphi$ if and only if $w \sqcap \mathcal{H}(\varphi) \models \varphi$.*

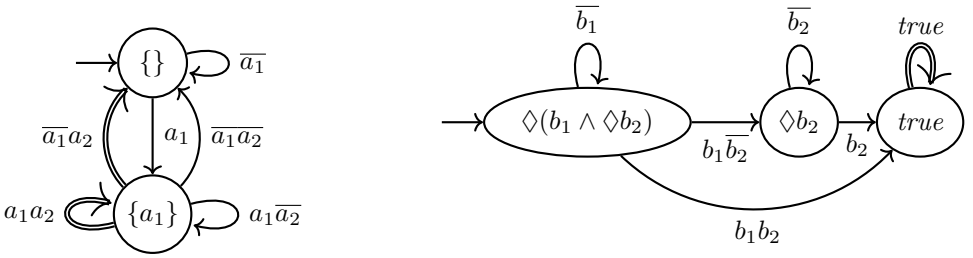
Proof. By induction on φ. For succinctness we just exhibit two cases and all other cases are analogous.
Case $\varphi = \bigcirc\psi$. Then, $w \models \varphi$ iff $w\,[1\ldots] \models \psi$ iff $w\,[1\ldots] \sqcap \mathcal{H}(\psi) \models \psi$ iff $\varnothing(w\,[1\ldots] \sqcap \mathcal{H}(\psi)) \models \varphi$ iff $w \sqcap \mathcal{H}(\varphi) \models \varphi$.
Case $\varphi = \psi \wedge \psi'$. Then, $w \models \varphi$ iff $w \models \psi \wedge w \models \psi'$ iff $w \sqcap \mathcal{H}(\psi) \models \psi \wedge w \sqcap \mathcal{H}(\psi') \models \psi'$ iff $w \sqcap (\mathcal{H}(\psi) \sqcup \mathcal{H}(\psi')) \models \varphi$ iff $w \sqcap \mathcal{H}(\varphi) \models \varphi$. □

The deterministic, transition-based Emerson-Lei automaton we are constructing keeps a buffer masked by $\mathcal{H}$. Intuitively, the automaton delays the decision whether φ holds by $n = |\mathcal{H}(\varphi)| - 1$ steps and then decides whether it holds true, instead of non-deterministically guessing the future and verifying this guess as done in standard LTL translations.

Definition 5.3. *Let φ be an LTL($\bigcirc$) formula over AP and let $n = \mathsf{max}(|\mathcal{H}(\varphi)|-1, 0)$. We then define a transition-based Emerson-Lei automaton for $\Box\Diamond\varphi$:*

$$\mathcal{A}(\Box\Diamond\varphi) = (Q, 2^{AP}, \delta, \varnothing^n, \mathrm{Inf}(Z))$$

$$
\begin{array}{lcl}
Q & = & \{w \in (2^{AP})^n \,:\, \forall i.\ w[i] \subseteq \mathsf{cl}(\mathcal{H}(\varphi))[i]\} \\
\delta(\sigma w, \sigma') & = & w\sigma' \sqcap \mathsf{drop}(\mathsf{cl}(\mathcal{H}(\varphi))) \quad \textit{for all } \sigma, \sigma' \in 2^{AP} \textit{ and } w \in (2^{AP})^{n-1} \\
Z & = & \{(w, \sigma, w') \in \delta \,:\, w\sigma\varnothing^\omega \models \varphi\}
\end{array}
$$



(a) $\psi_1 = \Box\Diamond(a_1 \wedge \bigcirc a_2)$ (b) $\psi_2 = \Diamond(b_1 \wedge \Diamond b_2)$

Figure 5.2: Automata for ψ_1 and ψ_2. Double-lined edges denote accepting transitions.

Observe that we must take the closure of $\mathcal{H}(\varphi)$ before intersecting with the buffer. Otherwise, we might lose information while propagating letters from the back to the front of the buffer. Further, we can always drop the last set of letters of the relevant history, since a transition-based acceptance is used. In the context of state-based acceptance this needs to be stored in the buffer as well.

Let us apply this construction to our example: $\Box\Diamond(a_1 \wedge \bigcirc a_2)$. First, we get $\mathcal{H}(a_1 \wedge \bigcirc a_2) = \{a_1\}\{a_2\}$. Second, since we always drop the last set of letters, we have $\mathsf{drop}(\mathcal{H}(a_1 \wedge \bigcirc a_2)) = \{a_1\}$ and $n = 1$. Thus, we obtain the Emerson-Lei automaton shown in Figure 5.2a, which is in fact a (transition-based) Büchi automaton.

Theorem 5.4. *Let φ be an LTL($\bigcirc$) formula over AP.*

$$L(\Box\Diamond\varphi) = L(\mathcal{A}(\Box\Diamond\varphi))$$

Proof. Assume $w \models \Box\Diamond\varphi$ holds. Thus, we have $\overset{\infty}{\exists}\, i.\ w[i\ldots] \models \varphi$ and we obtain $\overset{\infty}{\exists}\, i.\ w[i\ldots] \sqcap \mathcal{H}(\varphi) \models \varphi$ by using Lemma 5.2. Thus, there exists a finite word $w' \in (2^{AP})^*$ with (1) $w'\varnothing^\omega = w_i \sqcap \mathcal{H}(\varphi)$, (2) $w'\varnothing^\omega \models \varphi$, and (3) $|w'| = |\mathcal{H}(\varphi)|$. Thus, $\mathcal{A}(\Box\Diamond\varphi)$ infinitely often takes the (shortened) transition $t = (w'[0]\ldots w'[n-1], w'[n])$. Due to (2) we have $t \in \alpha$ and thus $w \in L(\mathcal{A}(\Box\Diamond\varphi))$. The other direction is analogous. □

Since $\Diamond\Box\varphi$ is equivalent to $\neg\Box\Diamond\neg\varphi$, we immediately obtain also a translation for $\mathrm{LTL}_{\Diamond\Box}(\bigcirc)$. We only need to change the acceptance condition to $\mathrm{Fin}\,(Z)$ with $Z = \{(w, \sigma, w') \in \delta : w\sigma\varnothing^\omega \not\models \varphi\}$.

5.1.3 Safety- and Cosafety-LTL

Translating safety LTL to deterministic automata is a well-studied problem. Since these languages can be defined using bad prefixes, meaning once a bad prefix has been read, the word is rejected, most automata generated by most available translations will have a single rejecting sink. All other states and transitions are then either rejecting or accepting. We use the straight-forward approach to apply the *af*-function

from [EKS16] to obtain a deterministic automaton for cosafety LTL formulas and by duality also for automata for safety languages. The *af*-function computes the left-derivative of a language expressed as an LTL formula:

$$
\begin{aligned}
af(\mathit{true},\sigma) &= \mathit{true} \\
af(\mathit{false},\sigma) &= \mathit{false} \\
af(a,\sigma) &= \begin{cases} \mathit{true} & \text{if } a \in \sigma \\ \mathit{false} & \text{if } a \notin \sigma \end{cases} \\
af(\neg a,\sigma) &= \neg af(a,\sigma)
\end{aligned}
\qquad
\begin{aligned}
af(\varphi \wedge \psi,\sigma) &= af(\varphi,\sigma) \wedge af(\psi,\sigma) \\
af(\varphi \vee \psi,\sigma) &= af(\varphi,\sigma) \vee af(\psi,\sigma) \\
af(\bigcirc\varphi,\sigma) &= \varphi \\
af(\Diamond\varphi,\sigma) &= af(\varphi,\sigma) \vee \Diamond\varphi \\
af(\varphi\,\mathcal{U}\,\psi,\sigma) &= af(\psi,\sigma) \vee (af(\varphi,\sigma) \wedge \varphi\,\mathcal{U}\,\psi)
\end{aligned}
$$

In the actual construction of $\mathcal{A}(\varphi)$, we consider the LTL formulas up to propositional equivalence. For the equivalence class of φ, we write $[\varphi]_P$. For example, $\mathit{true} \vee \Diamond a$ and true would be represented by the same state. Also, for an LTL formula φ of LTL($\mathcal{U}, \bigcirc$) we consider the set of equivalence classes of all subformulas, denoted by $\mathbf{S}(\varphi)$, instead of direct consideration of the subformulas.

Definition 5.5 ([EKS16], Definition 7)**.** *Let φ be a formula of LTL($\mathcal{U}, \bigcirc$), then*

$$\mathcal{A}(\varphi) = \Big(\mathbf{S}(\varphi), 2^{AP}, af, [\varphi]_P, \mathrm{Inf}\big(\{\mathit{true} \xrightarrow{\sigma} \mathit{true} \,:\, \sigma \in 2^{AP}\}\big)\Big).$$

Theorem 5.6 ([EKS16], Theorem 2)**.** *Let φ be a formula of LTL($\mathcal{U}, \bigcirc$), then*

$$L(\varphi) = L(\mathcal{A}(\varphi)).$$

For the cosafety formula $\Diamond(b_1 \wedge \Diamond b_2)$ we then obtain the automaton of Figure 5.2b with the accepting sink $[\mathit{true}]_P$. This approach also immediately tells us, when a run is accepting by looking at the state.

The handling of safety formulas $\varphi \in \mathrm{LTL}(\mathcal{R}, \bigcirc)$ can be done by the addition of the following rules for *af*:

$$
\begin{aligned}
af(\Box\varphi,\sigma) &= af(\varphi,\sigma) \wedge \Box\varphi \\
af(\varphi\,\mathcal{R}\,\psi,\sigma) &= af(\psi,\sigma) \wedge (af(\varphi,\sigma) \vee \varphi\,\mathcal{R}\,\psi)
\end{aligned}
$$

Additionally we change the acceptance condition to its dual version:

$$\mathrm{Fin}\big(\{\mathit{false} \xrightarrow{\sigma} \mathit{false} \,:\, \sigma \in 2^{AP}\}\big),$$

as a run ends in the trap state $[\mathit{false}]_P$, if it has consumed a bad prefix, and thus, the word is rejected.

5.1.4 General LTL

If the translation encounters a subformula not covered by Section 5.1.2 and Section 5.1.3, it resorts to an external general purpose LTL-to-deterministic-automaton

translation. Here the only restriction on the type of the automaton is that it has to be transition-based, since the translation for the fragments presented in Sections 5.1.2 and 5.1.3 produces transition-based Emerson-Lei automata. Nevertheless, converting state-based to transition-based acceptance can be easily achieved by shifting the acceptance from every state to its outgoing transitions. Also, every standard acceptance, like Büchi, Rabin, Streett or parity can be interpreted as Emerson-Lei acceptance.

5.1.5 Product construction

In the previous sections we discussed translations tailored to specific LTL fragments. Now we present a translation for full LTL. We split an LTL formula φ into several subformulas, that have a temporal operator on the root node in their syntax tree, and are not in the scope of another temporal operator. For all of such subformulas, we translate them into a deterministic automaton as described before, or if they do not fall into one of the supported fragments, we use a general LTL-to-deterministic-automata translation. Afterwards, all these deterministic automata yield a deterministic automaton for φ by a product construction. We first introduce the standard product construction for Emerson-Lei automata that is similar to the product construction for Muller automata and then move on to the enhanced construction.

Consider the following parametric formula: $\square\lozenge(a_1 \wedge \bigcirc(a_2 \wedge \ldots \bigcirc a_m)) \wedge \lozenge(b_1 \wedge \lozenge(b_2 \wedge \ldots \lozenge b_n))$. We will later demonstrate that the propagation of information allows us to construct a Büchi automaton of size $O(n+m)$, while `SPOT` in the standard configuration yields automata of size $O(n \cdot m)$ and only after enabling simulation-based reductions this decreases to sizes comparable to our automata. Let us now examine the construction, while we translate the formula $\square\lozenge(a_1 \wedge \bigcirc a_2) \wedge \lozenge(b_1 \wedge \lozenge b_2)$.

Standard Construction. For the product construction for an LTL formula φ we assume we are given a deterministic Emerson-Lei automaton $\mathcal{A}(\psi)$ for every $\psi \in \mathsf{sf}(\varphi)$. We denote by $Q^{\mathcal{A}(\psi)}$ the states of $\mathcal{A}(\psi)$ and by $q_0^{\mathcal{A}(\psi)}$ the initial state of $\mathcal{A}(\psi)$. Every state of the product automaton is a mapping from $\mathsf{sf}(\varphi)$ to $\bigcup_{\psi \in \mathsf{sf}(\varphi)} Q^{\mathcal{A}(\psi)}$ where every subformula ψ is mapped to a state in $Q^{\mathcal{A}(\psi)}$. We denote by $s[\psi] = q$ the current state of the automaton $\mathcal{A}(\psi)$ in the product state s, meaning $\psi \mapsto q \in s$.

Definition 5.7. *Let φ be a formula and for every $\psi \in \mathsf{sf}(\varphi)$ let $\mathcal{A}(\psi)$ be a deterministic Emerson-Lei automaton recognizing $L(\psi)$. The deterministic Emerson-Lei automaton for the product is defined as:*

$$\mathcal{A}^{\times}(\varphi) = (Q, 2^{AP}, \delta, q_0, A(\varphi))$$

$$\delta(s,\sigma) = \{\psi \mapsto \delta^{\mathcal{A}(\psi)}(s[\psi],\sigma) \,:\, \psi \in \mathsf{sf}(\varphi)\} \qquad q_0 = \{\psi \mapsto q_0^{\mathcal{A}(\psi)} \,:\, \psi \in \mathsf{sf}(\varphi)\}$$

Further, Q is defined as the set of all reachable states. The acceptance condition is recursively computed over the structure of φ with $\uparrow$ denoting the lifting of the acceptance condition:

$$\begin{aligned} A(\mathit{true}) &= \mathit{true} & A(\varphi \wedge \psi) &= A(\varphi) \wedge A(\psi) \\ A(\mathit{false}) &= \mathit{false} & A(\varphi \vee \psi) &= A(\varphi) \vee A(\psi) \end{aligned}$$

$$\alpha(\psi) \quad = \quad \uparrow \alpha^{\mathcal{A}(\psi)} \text{ for } \psi \in \mathsf{sf}(\varphi)$$

Since all $\delta^{\mathcal{A}}(\psi)$ are deterministic, δ is also deterministic.

Theorem 5.8. *Let φ be an LTL formula. Then*

$$L(\varphi) = L(\mathcal{A}^{\times}(\varphi))$$

Enhanced Construction. An essential part of the enhanced product construction is the removal of unnecessary information from the product states. For this we introduce three additional states with special semantics: q_{acc} signalizes that the component moved to an accepting sink state, while q_{rej} expresses that the component moved to a rejecting sink state. Alternatively, if a component got irrelevant for the acceptance condition it is also moved to q_{rej}. Lastly, q_{hold} says that the component was put on hold. More specifically, we put the fairness automata on hold, if a "neighboring" automaton still needs to fulfill its goal, such as reaching an accepting trap. To make notation easier to read we assume that every automaton $\mathcal{A}(\varphi)$ contains these states and all accepting sinks (or traps) have been replaced by q_{acc} and rejecting by q_{rej}. We use the following abbreviations to reason about LTL formulas:

- $\mathsf{conj}(\varphi)$ ($\mathsf{disj}(\varphi)$) denotes the set of all conjuncts of a conjunction (disjuncts of a disjunction) outside the scope of a temporal operator, e.g. let $\varphi = \Diamond a \wedge (\bigcirc b \vee \Box c)$, then $\mathsf{conj}(\varphi) = \{\{\Diamond a, \bigcirc b \vee \Box c\}\}$ and $\mathsf{disj}(\varphi) = \{\{\bigcirc b, \Box c\}\}$.
- $\varphi[\mathcal{F} \leftarrow \psi]$ denotes the substitution of all formulas in the set $\mathcal{F}$ with the formula ψ, e.g. $(\Diamond a \wedge (\bigcirc b \vee \Box c))[\{\Diamond a, \Box a\} \leftarrow \mathit{true}] = \mathit{true} \wedge (\bigcirc b \vee \Box c)$.
- $\mathsf{support}(\varphi)$ denotes the support of a formula, where the formula is viewed as a propositional formula, which means that temporal operators are also considered propositions, e.g., $\mathsf{support}((\bigcirc a \wedge \Diamond b) \vee (\Diamond b)) = \{\Diamond b\}$. This means every assignment can be restricted to the propositions of the support: $S \models_P \varphi \Leftrightarrow S \cap \mathsf{support}(\varphi) \models_P \varphi$, where $\models_P$ denotes the conventional propositional satisfaction relation.

We use the following definitions to manipulate product states:

Definition 5.9 (Product state modifications)**.** *An* update *of a product state tests a predicate P on a formula-state pair (ψ, q) and replaces q with a new value obtained by the updater U depending on ψ if it holds:*

$$\mathsf{update}(s, P, U) = \{\psi \mapsto (\textbf{\textit{if}}\ P(\psi, q)\ \textbf{\textit{then}}\ U(\psi)\ \textbf{\textit{else}}\ q)\ :\ \psi \mapsto q \in s\}$$

$\mathsf{prune}(s)$ *disables automata in* s *that became irrelevant for the acceptance condition, meaning there are no longer in the support of the original formula after using knowledge from other automata. For this let us denote by* $\mathcal{F}_{\mathsf{acc}}$ *all* $\psi \mapsto q_{\mathsf{acc}} \in s$ *and by* $\mathcal{F}_{\mathsf{rej}}$ *all* $\psi \mapsto q_{\mathsf{rej}} \in s$.

$$\begin{aligned} \mathsf{prune}(s) &= \mathsf{update}(s, P, U) \\ P(\psi, q) &= (q \neq q_{\mathsf{acc}} \wedge \psi \notin \mathsf{support}(\varphi[\mathcal{F}_{\mathsf{acc}} \leftarrow \mathit{true}, \mathcal{F}_{\mathsf{rej}} \leftarrow \mathit{false}])) \\ U(\psi) &= q_{\mathsf{rej}} \end{aligned}$$

start (s) *starts (fairness) automata that are required for the acceptance but have been put on hold. This is the case, if automata with terminal acceptance for formulas in the same conjunction (start* $(s)_c$*) have not yet reached* q_{acc} *or the dual case for disjunctions:*

$$\begin{aligned} \mathit{start}\,(s)_c &= \mathsf{update}(s, P_c, U) \\ \mathit{start}\,(s)_d &= \mathsf{update}(s, P_d, U) \\ P_c(\psi, q) &= (q = q_{\mathsf{hold}} \wedge \exists C \in \mathsf{conj}(\varphi).\ \psi \in C \wedge \forall \chi \in C \cap \mathit{LTL}(\mathcal{U}, \bigcirc).\ s[\chi] = q_{\mathsf{acc}}) \\ P_d(\psi, q) &= (q = q_{\mathsf{hold}} \wedge \exists D \in \mathsf{disj}(\varphi).\ \psi \in D \wedge \forall \chi \in D \cap \mathit{LTL}(\mathcal{R}, \bigcirc).\ s[\chi] = q_{\mathsf{rej}}) \\ U(\psi) &= q_0^{\mathcal{A}(\psi)} \end{aligned}$$

Definition 5.10 (Enhanced product automaton)**.** *Let* φ *be a formula. The Emerson-Lei automaton for the enhanced product automaton is defined the same way as Definition 5.7 with the following changes:*

$$\mathcal{A}_E^{\times}(\varphi) = (Q, 2^{AP}, \delta, q_0, A(\varphi))$$

$$\begin{aligned} \delta(s, \sigma) &= \mathit{start}\left(\mathsf{prune}(\{\psi \mapsto \delta^{\mathcal{A}(\psi)}(s[\psi], \sigma) \,:\, \psi \in \mathsf{sf}(\varphi)\})\right) \\ q_0 &= \mathit{start}\left(\left\{\psi \mapsto \begin{cases} q_0^{\mathcal{A}(\psi)} & \mathit{if}\ \ \psi \in \mathsf{sf}(\varphi) \setminus \mathit{LTL}_{\Diamond\Box,\Box\Diamond}(\bigcirc) \\ q_{\mathsf{hold}} & \mathit{otherwise} \end{cases}\right\}\right) \end{aligned}$$

Theorem 5.11. *Let* φ *be a formula.*

$$L(\varphi) = L(\mathcal{A}_E^{\times}(\varphi))$$

If we apply this construction to $\Box\Diamond(a_1 \wedge \bigcirc a_2) \wedge \Diamond(b_1 \wedge \Diamond b_2)$, we obtain the automaton shown in Figure 5.3. Observe that $\Box\Diamond(a_1 \wedge \bigcirc a_2)$ is put on hold until the automaton for $\Diamond(b_1 \wedge \Diamond b_2)$ reaches q_{acc}.

Further Optimizations

There are two further optimizations we implement: First, we replace the local histories of each automaton for $\mathrm{LTL}_{\Diamond\Box,\Box\Diamond}(\bigcirc)$ with one *global* history. Second, we *piggyback* the acceptance of (co-)safety automata on neighboring fairness automata. Let $C \in \mathsf{conj}(\varphi)$ be a conjunction, $\psi_r \in \mathrm{LTL}(\mathcal{U}, \bigcirc) \cap C$ and $\psi_f \in \mathrm{LTL}_{\Diamond\Box}(\bigcirc) \cap C$. We then have $\alpha^{\mathcal{A}(\psi_f)} = \mathrm{Fin}\,(S)$ and extend S with $Q^{\mathcal{A}(\psi_r)} \setminus \{q_{\mathsf{acc}}\}$. The same trick can be applied to $\psi_f \in \mathrm{LTL}_{\Box\Diamond}(\bigcirc)$ and of course to the dual case with $\psi_s \in \mathrm{LTL}(\mathcal{R}, \bigcirc)$.

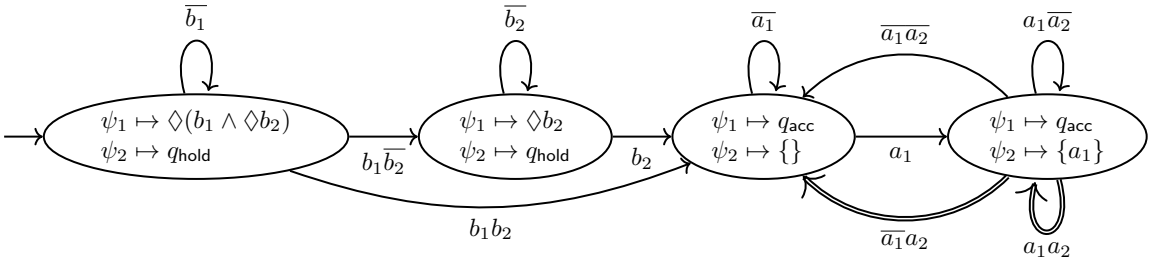


Figure 5.3: Enhanced product automaton for $\square\Diamond(a_1 \wedge \bigcirc a_2) \wedge \Diamond(b_1 \wedge \Diamond b_2)$, only the accepting edges for $\square\Diamond(a_1 \wedge \bigcirc a_2)$ are drawn.

5.2 End-component analysis for Emerson-Lei acceptance

The previous section dealt with the construction of deterministic Emerson-Lei automata. Now we detail the application of them in PMC. For this we assume that a product of the MDP and EL automaton has been already built in the standard way, and we start with a (product) Markovian model and an Emerson-Lei acceptance condition. In this section we assume, that every Emerson-Lei acceptance is transition-based, but analogous results can be stated for state-based acceptance. At first, we look at the easy case of a Markov chain.

Markov Chains. Markov chains can be checked efficiently against Emerson-Lei acceptance conditions: Assume a Markov chain $\mathcal{M}$ and an Emerson-Lei acceptance Φ. To calculate $\Pr_{\mathcal{M}}(\Phi)$ one has to check, for all reachable BSCCs, whether they are accepting or not. As almost surely every path ends in a BSCC and visits every state in it infinitely often, a BSCC is accepting iff the set of all states (or transitions) is accepting. To finish calculating $\Pr_{\mathcal{M}}(\Phi)$ one can solve the linear equation system of Figure 2.3. The algorithmic solution for the positivity problem, i.e., $\Pr_{\mathcal{M}}(\Phi) > 0$ immediately follows:

Lemma 5.12. *Deciding* $\Pr_{\mathcal{M}}(\Phi) > 0$ *for a Markov chain* $\mathcal{M}$ *and an Emerson-Lei acceptance* Φ *can be done in polynomial time.*

Of course, one can also decide $\Pr_{\mathcal{M}}(\Phi) > 0$ by pure graph-theoretic means, i.e., checking, whether there exists a reachable accepting BSCC.

5.2.1 Markov decision processes

Given an MDP $\mathcal{M} = (S, Act, P, \iota, AP, \ell)$ and an Emerson-Lei acceptance $\Phi \in \mathcal{C}_\delta$, the positivity problem is to decide whether $\Pr^{\max}_{\mathcal{M}}(\Phi) > 0$ holds.

Now we establish the connection between the satisfiability problem for Boolean formulas and the positivity problem. For that we show how to translate a Boolean

formula φ into an Emerson-Lei acceptance Φ. W.l.o.g. we assume φ to be given in positive normal form. For every atomic proposition x occurring in φ we construct an $\mathrm{Inf}\left(s \xrightarrow{x} s\right)$, where $s \xrightarrow{x} s$ is meant as a transition from state s to itself with action x, and $\mathrm{Inf}\left(s \xrightarrow{x} s\right)$ a short form notation for $\mathrm{Inf}\left(\{s \xrightarrow{x} s\}\right)$. Analogously, $\neg x$ is transformed into $\mathrm{Fin}\left(s \xrightarrow{x} s\right)$.

With this transformation, the NP-hardness for the satisfiability of a class of Boolean formulas results in the NP-hardness for the corresponding Emerson-Lei acceptance problem.

Theorem 5.13. *Let $\mathcal{C}$ be a class of Boolean formulas and $\mathcal{C} - \mathsf{SAT}$ the class of satisfiable Boolean formulas contained in $\mathcal{C}$. If $\mathcal{C} - \mathsf{SAT}$ is NP-complete, then deciding whether the positivity problem for MDPs and $\mathcal{C}$-Emerson-Lei acceptances is NP-complete as well.*

Proof. For proving the hardness we provide a polynomial reduction from $\mathcal{C} - \mathsf{SAT}$. Let φ be a $\mathcal{C}$-formula with atomic propositions $X = \{x_1, \ldots, x_n\}$. We generate a one-state MDP $\mathcal{M}$ with an x_i action and a transition $s \xrightarrow{x_i,1} s$ for every atomic proposition x_i in φ.

For further considerations, we identify a transition by its action, i.e., a transition subset $Y \subseteq \{s\} \times X \times \{1\} \times \{s\}$ can be identified by the corresponding labelings $\left\{x \in X \,:\, s \xrightarrow{x,1} s \in Y\right\}$. The acceptance condition Φ is obtained from φ by replacing every occurrence of x_i with $\mathrm{Inf}\,(x_i)$ and every occurrence of $\neg x_i$ with $\mathrm{Fin}\,(x_i)$.

Obviously, a transition subset $Y \subseteq X$ (seen as a set of actions) satisfies Φ if and only if $Y \models \varphi$. Every model of φ can be transformed to a corresponding end-component of $\mathcal{M}$ by visiting Y infinitely often, since there is only one state with a self-loop for every action $x \in X$. Analogously, every subset of transitions Y satisfying the acceptance condition Φ is also a model for φ. Therefore, $\mathcal{M}$ is positive for Φ if and only if φ is satisfiable.

To prove that the positivity problem belongs to NP, we assume that we are given an MDP $\mathcal{M}$ and an Emerson-Lei acceptance Φ. One can choose non-deterministically an end-component, and check whether the set of transitions in the end-component satisfies Φ. □

The proof of Theorem 5.13 can be modified to work with state-based acceptance. The key idea is to introduce a state for every atomic proposition in φ and to identify the state space with the atomic propositions.

Clearly, $\mathrm{Inf}\,(A) \vee \mathrm{Inf}\,(B) \equiv \mathrm{Inf}\,(A \cup B)$. Hence, each CNF acceptance formula can be transformed into an equivalent generalized Streett formula of the form

$$\bigwedge_{1 \leq i \leq n} \mathrm{Fin}\,(A_{i,1}) \vee \ldots \vee \mathrm{Fin}\,(A_{i,k_i}) \vee \mathrm{Inf}\,(B_i) \equiv \bigwedge_{1 \leq i \leq n} \left(\mathrm{Inf}\left(\overline{A_{i,1}}\right) \wedge \ldots \wedge \mathrm{Inf}\left(\overline{A_{i,k}}\right) \rightarrow \mathrm{Inf}\,(B_i)\right)$$

Lemma 5.14. *Let $\mathcal{M}$ be an MDP and Φ a generalized Streett acceptance. Then deciding whether $\mathcal{M}$ is positive for Φ is NP-complete.*

Note, that the class of generalized Streett acceptance corresponds to the class of Horn formulas. So, the reverse direction of Theorem 5.13 does not hold, unless $P = NP$. Another prominent example class $\mathcal{C}$ of propositional CNF-formulas where the satisfiability problem can be solved in polynomial time is 2CNF. The full class of 2CNF acceptance for deciding non-emptiness in an automaton is NP-complete, see Lemma 5.15.

Let $\mathsf{2CNF} - \mathsf{NEG}$ denote the set of all 2CNF-formulas where only negative literals occur, e.g., $(\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_4)$. All $\mathsf{2CNF} - \mathsf{NEG}$ formulas are satisfiable, since the valuation that assigns *false* to every atomic proposition is a model for every $\mathsf{2CNF} - \mathsf{NEG}$ formula. However, the corresponding problem to check whether an MDP is positive for a $\mathsf{2CNF} - \mathsf{NEG}$ acceptance is NP-complete. An analogous NP-completeness result for checking emptiness of an automaton with a $\mathsf{2CNF} - \mathsf{NEG}$ acceptance has been proven by Emerson and Lei, see [EL87]. This proof can be adapted in a straight-forward manner for MDPs.

Lemma 5.15 (see Theorem 4.7 of [EL87])**.** *Deciding non-emptiness for an automaton with a $\mathsf{2CNF} - \mathsf{NEG}$ acceptance condition is NP-complete.*

Despite the NP-hardness of 2CNF acceptance, several subclasses of 2CNF acceptance are solvable in polynomial time. So, the positivity problem for Streett acceptance [BGC09b] and clearly Rabin acceptance is decidable in polynomial time (see Section 2.4.1).

In general, deciding positivity of $\mathcal{M}$ (and Φ) can be reduced to the search of some reachable EC satisfying Φ:

By Lemma 2.6 we know that $\Pr^{\max}_{\mathcal{M}}(\Phi) > 0$ iff there exists some reachable EC $\mathcal{T} = (T, A)$ such that Φ is satisfied by all transitions occurring in $\mathcal{T}$.

What remains to explain is how to check whether a particular EC satisfies Φ. For this, we take the set of transitions of the EC (here we denote it by Δ) and set every $\mathrm{Inf}(Z)$ in Φ to *true* if $Z \cap \Delta \neq \varnothing$, otherwise to *false*. Analogously, we set $\mathrm{Fin}(Z)$ to *true* if $Z \cap \Delta = \varnothing$, otherwise to *false*. The overall evaluation of Φ for EC follows immediately in the standard way.

Algorithm 1 implements a naive positivity check for a given MDP $\mathcal{M}$ and Emerson-Lei acceptance Φ. It enumerates all reachable ECs and checks whether there is an EC $\mathcal{T} = (T, A)$ satisfying Φ. For the last step we take a function CheckEC for granted, which can be implemented as explained above. As there may be an exponential number of ECs, Algorithm 1 may require exponential time.

5.2.2 $\mathrm{Fin}(\cdot)$-less acceptance

If the acceptance condition Φ does not use any $\mathrm{Fin}(\cdot)$ operator, for any transition sets T and T' with $T \subseteq T'$ we have $T \models \Phi \implies T' \models \Phi$, i.e., visiting more transitions cannot invalidate the acceptance condition. This implies that for checking

Algorithm 1 Checking $\Pr^{\max}_{\mathcal{M}}(\Phi) > 0$ amounts to showing that there is a reachable EC satisfying Φ.

```
1: function Check(MDP M, EL acceptance Φ)
2:     for all T ∈ ECDecomp(M) do
3:         if CheckEC(T, Φ) then
4:             Return true
5:         end if
6:     end for
7:     Return false
8: end function
```

positiveness of MDPs with $\mathrm{Fin}(\cdot)$-less acceptance, it is enough to check whether any maximal EC $\mathcal{T}$ satisfies Φ.

Algorithm 2 Implementation of Check for $\mathrm{Fin}(\cdot)$-less acceptance.

```
1: function Check(MDP M, EL acceptance Φ)
2:     for all T ∈ MECDecomp(M) do
3:         Φ' ← cut(Φ, T)
4:         if Φ' ≠ false then
5:             Return true
6:         end if
7:     end for
8:     Return false
9: end function
```

Algorithm 2 shows an implementation of this approach. Similar to Algorithm 1 it searches for an accepting end-component, but here it suffices to analyze the maximal end-components only. Therefore we use the MEC decomposition function called MECDecomp instead of ECDecomp. Algorithm 2 also relies on a function cut(Φ, Δ) for a transition based EL acceptance Φ and a transition set Δ. This function replaces every occurrence of $\mathrm{Inf}(Z)$ with $\mathrm{Inf}(Z \cap \Delta)$. We will reuse cut$(\Phi, \Delta)$ in the following section, and there also every occurrence of $\mathrm{Fin}(Z)$ is replaced with $\mathrm{Fin}(Z \cap \Delta)$ for every set Z.

We overload cut(Φ, Δ) by cut(Φ, $\mathcal{T}$) for an end-component $\mathcal{T}$, where we identify $\mathcal{T}$ by its transitions Δ.

Also, after the previous replacements, the following rewrite rules (and if applicable, their symmetric rules) are applied:

$$\begin{aligned} \mathrm{Inf}(\varnothing) &\mapsto \mathit{false} & \mathrm{Fin}(\varnothing) &\mapsto \mathit{true} \\ \Phi \wedge \mathit{false} &\mapsto \mathit{false} & \Phi \wedge \mathit{true} &\mapsto \Phi \\ \Phi \vee \mathit{false} &\mapsto \Phi & \Phi \vee \mathit{true} &\mapsto \mathit{true} \end{aligned}$$

5.3 A DPLL based positivity check

As Theorem 5.13 suggests, there is a close connection between Emerson-Lei acceptances and Boolean formulas. We can see Emerson-Lei acceptances as Boolean formulas in positive normal form with $\mathrm{Inf}(\cdot)$ as positive Boolean variables and $\mathrm{Fin}(\cdot)$ as negative Boolean variables. One of the most important decision problems in the context of Boolean formulas is satisfiability, called SAT. Among SAT solving algorithms, one of the most prominent ones is the DPLL algorithm, introduced by Davis, Putman, Logemann, and Loveland [DP60; DLL62]. It relies on a split on the truth values of variables enhanced with heuristics such as unit rule and pure literal rule.

In this section we investigate the connection between Boolean formulas and Emerson-Lei acceptance further and present a DPLL-based `Check` procedure for deciding positivity of an MDP.

5.3.1 Tseytin transformation for Emerson-Lei Acceptance

Since the DPLL algorithm presumes conjunctive normal form (CNF) for a propositional formula, we use an adapted Tseytin transformation [Tse68]. Let us first introduce some helpful notations. Let Φ be an Emerson-Lei acceptance. We call Φ Boolean if it is of the form $\Psi \wedge \Gamma$ or $\Psi \vee \Gamma$. The set of the Boolean subformulas of Φ is denoted by $\mathbf{B}(\Phi)$.

For the Tseytin transformation we introduce a slack variable s_Ψ for every Boolean subformula $\Psi \in \mathbf{B}(\Phi)$. We denote the set of slack variables by $\mathbf{V}$. The function $s : \mathcal{C}_\delta \to \mathbf{V} \cup AP$ maps every Boolean Emerson-Lei acceptance to its corresponding slack variable and preserves atomic propositions:

$$s(\Phi) = \begin{cases} s_\Phi & \text{if } \Phi \text{ is Boolean} \\ \Phi & \text{otherwise} \end{cases}$$

The idea of the Tseytin transformation is to introduce a slack variable for every Boolean subformula, signaling whether the subformula should be true or not. Therefore, the function t introduces a slack variable for a Boolean subformula and transforms the subformula into a clause.

$$t(\Phi) = \begin{cases} (\neg s(\Phi) \vee s(\Psi)) \wedge (\neg s(\Phi) \vee s(\Gamma)) & \text{if } \Phi = \Psi \wedge \Gamma \\ \neg s(\Phi) \vee s(\Psi) \vee s(\Gamma) & \text{if } \Phi = \Psi \vee \Gamma \\ \Phi & \text{otherwise} \end{cases}$$

As a whole, we get the following satisfiability-equivalent formula $\mathsf{cnf}(\Phi)$ in conjunctive normal form for an Emerson-Lei acceptance Φ:

$$\mathsf{cnf}(\Phi) = s(\Phi) \wedge \bigwedge_{\Psi \in \mathbf{B}(\Phi)} t(\Psi)$$

As a heuristic we allow the consolidation of binary conjunction to n-ary conjunctions. For example, in the generalized Rabin acceptance

$$(\mathrm{Fin}\,(U_1) \wedge \mathrm{Inf}\,(L_{1,1}) \wedge \mathrm{Inf}\,(L_{1,2})) \vee (\mathrm{Fin}\,(U_2) \wedge \mathrm{Inf}\,(L_{2,1}) \wedge \mathrm{Inf}\,(L_{2,2}))$$

both generalized Rabin pairs would be summarized to one conjunction each.

5.3.2 DPLL-based positivity check

For deciding positivity of an MDP $\mathcal{M}$ and an Emerson-Lei acceptance Φ, our DPLL-based algorithm analyzes every reachable MEC one after another. If an MEC containing a satisfying EC is found, then $\Pr^{\max}_{\mathcal{M}}(\Phi)$ is positive. In contrast to the naive approach of Algorithm 1 the method here does not explicitly enumerate all ECs, but rather chooses slack variables or $\mathrm{Fin}\,(\cdot)$ variables, and in case of $\mathrm{Fin}\,(\cdot)$ variables, prunes the MEC and enumerate all remaining MECs in the pruned MEC.

In contrast to purely Boolean formulas, we have in case of Emerson-Lei acceptance $\mathrm{Fin}\,(\cdot)$-variables, $\mathrm{Inf}\,(\cdot)$-variables, and, additionally introduced by the Tseytin transformation, Boolean slack variables. This leads to a different branching behavior. During the recursion, our adaption of the DPLL algorithm always considers end-components. In this sense, an $\mathrm{Inf}\,(\cdot)$ that shares transitions with the EC can be easily satisfied. We can thus assume that every $\mathrm{Inf}\,(\cdot)$ with a non-empty transition set is currently *true*, since by visiting every transition in the current end-component, every $\mathrm{Inf}\,(\cdot)$ variable becomes *true*. So, we branch only over $\mathrm{Fin}\,(\cdot)$, and slack variables. Still, during the recursion, an $\mathrm{Inf}\,(\cdot)$ can become *false* if every transition that is required for the satisfaction of the $\mathrm{Inf}\,(\cdot)$ becomes pruned.

The complete algorithm is depicted in Algorithm 4 (relying on the helper functions depicted in Algorithm 3). The core method is the recursive function CheckMEC($\mathcal{T}, \Phi \in \mathcal{C}_\delta$). It works on a particular MEC of $\mathcal{M}$ and decides, whether the Emerson-Lei acceptance condition is satisfied within the MEC or not. It starts with a precalculation (see function Precalculation in Algorithm 3), where every $\mathrm{Fin}\,(\cdot)$- and every $\mathrm{Inf}\,(\cdot)$-set is intersected with the transitions that are completely inside the currently analyzed MEC. Then, we check for the easy cases. Every atomic proposition $\mathrm{Inf}\,(\varnothing)$ is unsatisfiable and can be removed from the formula. Likewise, every $\mathrm{Fin}\,(\varnothing)$ is always satisfied (in a non-empty MEC), which allows us to remove every clause with a $\mathrm{Fin}\,(\varnothing)$ literal. Afterwards, we test $\mathcal{T}$ for emptiness and return *false* if this is the case, as the emptiness precludes the existence of any cycle. Analogously, we check $\Phi = \mathit{false}$ after the application of Cut. We also check whether every clause contains an $\mathrm{Inf}\,(\cdot)$ variable. If this is the case, every clause can be made true easily by visiting every transition of $\mathcal{T}$ infinitely often and we return *true*.

After precalculation, we apply two heuristics: the unit clause rule and the pure literal rule. The first checks whether there exists a clause with exactly one literal of the form s, $\neg s$, or $\mathrm{Fin}\,(Z)$. If there exists such a clause, the corresponding slack variable must be set to a truth value such that the clause evaluates to *true*.

Algorithm 3 Helper functions for Algorithm 4.

function PRECALCULATION(MEC $\mathcal{T}$, EL acceptance Φ)
    $\Phi \leftarrow \text{CUT}(\Phi, \mathcal{T})$
    if $\mathcal{T}$ is empty or $\Phi = \mathit{false}$ **then**
        Return *false*
    end if
    if every clause contains an $\text{Inf}(\cdot)$ **then**
        Return *true*
    end if
    Return Φ
end function

function BRANCH(EC $\mathcal{T} = (T, A)$, $\Phi \in \mathcal{C}_\delta$, $l \in \mathbf{V} \cup AP$, $v \in \{\mathit{true}, \mathit{false}\}$)
    for all $\mathcal{T}' \in \text{UPDATE}(\mathcal{T}, l, v)$ **do**
        if CHECKMEC$(\mathcal{T}', \Phi[l \leftarrow v])$ **then**
            Return *true*
        end if
    end for
    Return *false*
end function

function UPDATE(MEC $\mathcal{T} = (T, A)$, $l \in \mathbf{V} \cup AP$, $v \in \{\mathit{true}, \mathit{false}\}$)
    $\mathcal{T}' \leftarrow \{\mathcal{T}\}$
    if $(l = \text{Fin}(Z)) \wedge (v = \mathit{true})$ **then**
        $\mathcal{T}' \leftarrow \text{MECDECOMP}(\mathcal{T} \setminus Z)$
    end if
    Return T'
end function

Algorithm 4 A DPLL based algorithm for checking the satisfiability of a Emerson-Lei acceptance Φ in CNF within a given end-component.

```
1: function CheckMEC(MEC 𝒯, EL acceptance Φ ∈ 𝒞_δ)
2:     Φ ← precalculation(𝒯, Φ)
3:     if Φ ∈ {true, false} then
4:         Return Φ
5:     end if
6:     //Unit rule
7:     if Φ contains a unit clause κ of the form Fin(Z) or s then
8:         Return Branch(𝒯, Φ, κ, true)
9:     end if
10:    if Φ contains a unit clause κ of the form ¬s then
11:        Return Branch(𝒯, Φ, s, false)
12:    end if
13:    //Pure literal rule
14:    if Φ contains a slack variable l and the complement does not occur then
15:        Return Branch(𝒯, Φ, l, true)
16:    end if
17:    if Φ contains a slack variable ¬l and the complement does not occur then
18:        Return Branch(𝒯, Φ, l, false)
19:    end if
20:    //Branching
21:    κ ← chooseLiteral(Φ)
22:    Return Branch(𝒯, Φ, κ, true) ∨ Branch(𝒯, Φ, κ, false)
23: end function

24: function Check(MDP ℳ, EL acceptance Φ)
25:    for all 𝒯 ∈ MECDecomp(ℳ) do
26:        if CheckMEC(𝒯, Φ) then
27:            Return true
28:        end if
29:    end for
30:    Return false
31: end function
```

The pure literal rule is applicable if there exists a slack variable occurring solely positive, or solely negative in Φ. Again, we do not need to split on the truth values of the slack variable, since we can assume safely that this literal evaluates *true*.

If none of the two heuristics is applicable, we choose a literal (either one of the slack literals or a $\mathrm{Fin}(\cdot)$ literal) and branch over it. We search for a satisfying assignment by setting the literal at first to be *true*, and afterwards to be *false*.

Every time we set a $\mathrm{Fin}(Z)$ variable to *true*, we have to remove every transition in Z from our current maximal end-component $\mathcal{T}$. Since this may violate the requirements for an end-component, we enumerate all MECs contained in $\mathcal{T} \setminus Z$ and branch into them. This ensures that the algorithm returns *true* iff there exists an end-component fulfilling the acceptance.

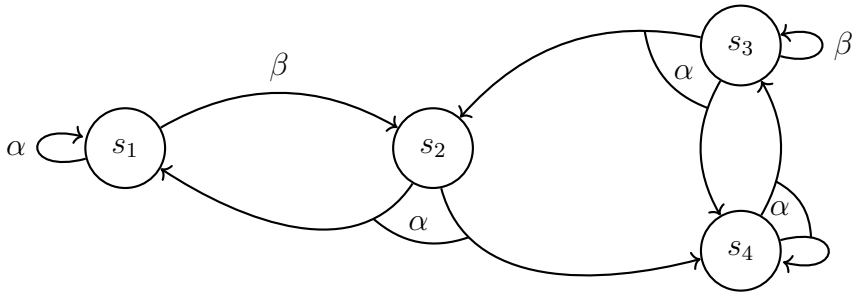


Figure 5.4: A maximal end-component with six end-components contained in it. In particular, the end-components $(\{s_1\}, \{s_1 \mapsto \{\alpha\}\})$ and $(\{s_3\}, \{s_3 \mapsto \{\beta\}\})$ among them.

Example 5.16. *Let*

$$\Phi_0 = \Big(\neg x \vee \mathrm{Fin}\big(s_1 \xrightarrow{\alpha} s_1\big) \vee \mathrm{Fin}\big(s_2 \xrightarrow{\alpha} s_1, s_2 \xrightarrow{\alpha} s_4\big)\Big) \wedge x \wedge \Big(\neg x \vee \mathrm{Fin}\big(s_4 \xrightarrow{\alpha} s_3\big) \vee \mathrm{Inf}\big(s_3 \xrightarrow{\beta} s_3\big)\Big)$$

be an Emerson-Lei acceptance after the Tseytin transformation and $\mathcal{T}$ be the MEC depicted in Figure 5.4. We first apply the unit rule for x and obtain

$$\Phi_1 = \Big(\mathrm{Fin}\big(s_1 \xrightarrow{\alpha} s_1\big) \vee \mathrm{Fin}\big(s_2 \xrightarrow{\alpha} s_1, s_2 \xrightarrow{\alpha} s_4\big)\Big) \wedge \Big(\mathrm{Fin}\big(s_4 \xrightarrow{\alpha} s_3\big) \vee \mathrm{Inf}\big(s_3 \xrightarrow{\beta} s_3\big)\Big).$$

Assume that variable $\mathrm{Fin}\big(s_2 \xrightarrow{\alpha} s_1, s_2 \xrightarrow{\alpha} s_4\big)$ *is chosen as branching literal, and initially considered to be true. This leads to two possible MECs to branch in:* $(\{s_1\}, \{s_1 \mapsto \{\beta\}\})$ *and* $(\{s_3\}, \{s_3 \mapsto \{\beta\}\})$.

In Φ_1 *the first clause is removed, thus*

$$\Phi_2 = \mathrm{Fin}\big(s_4 \xrightarrow{\alpha} s_3\big) \wedge \mathrm{Inf}\big(s_3 \xrightarrow{\beta} s_3\big)$$

remains. Still, it needs to be checked, whether one of the two MECs satisfies Φ_2.

In case of $(\{s_1\}, \{s_1 \mapsto \{\beta\}\})$ *the acceptance simplifies to* $\mathrm{Fin}(\varnothing) \vee \mathrm{Inf}(\varnothing)$ *and therefore true. In case of* $(\{s_3\}, \{s_3 \mapsto \{\beta\}\})$ *the acceptance is reduced to* $\mathrm{Inf}\big(s_3 \xrightarrow{\beta} s_3\big)$ *but now in every clause there is an* $\mathrm{Inf}(Z)$ *with* $Z \neq \varnothing$*, so the algorithm returns true.*

Emerson-Lei automata. Deciding non-emptiness for automata profits from similar techniques as probabilistic model checking.

We can see an Emerson-Lei automaton as a Markov decision process where every transition probability equals 1. So, deciding non-emptiness for Emerson-Lei automata reduces to the search for an accepting cycle. Therefore, we analyze every SCC as described in Algorithm 4, but with SCC decomposition instead of MEC decomposition.

To sum up, we can formulate Theorem 5.13 also for Emerson-Lei automata. Since non-emptiness and emptiness are dual to each other, we stick to the usual formulation of emptiness checking.

Lemma 5.17 (see Theorem 5.13)**.** *Let $\mathcal{C} - \mathsf{SAT}$ be a class of Boolean formulas, for which the satisfiability problem is NP-complete. Then, deciding emptiness $\mathcal{L}_\omega(\mathcal{A}) = \varnothing$ for an Emerson-Lei automaton $\mathcal{A}$ with an acceptance condition Φ in $\mathcal{C}$ is coNP-complete.*

5.3.3 Choosing the branching literal

In Algorithm 4 the function chooseLiteral is left unspecified. In fact, any function returning a literal (from the slack or $\mathrm{Fin}(\cdot)$ literals) that occurs in Φ leads to a correct implementation. To achieve a good performance in practice it is interesting to consider several heuristics to select a beneficial literal for the branching step.

One important rule is that we prefer to branch over slack literals if there are any and only if no slack literal is left we branch over $\mathrm{Fin}(\cdot)$ literals. This may lead to faster pruning of clauses, since every slack variable stands for a whole subformula of the original formula. Especially in case of a generalized Rabin acceptance this leads to a polynomial pair-by-pair check.

5.3.4 Generalized Rabin acceptance

Tseytin transformation for generalized Rabin acceptance. The generalized version of a Rabin condition is a disjunction of conjunctions of a $\mathrm{Fin}(\cdot)$ variable and several $\mathrm{Inf}(\cdot)$ variables:

$$\Phi = \big(\mathrm{Fin}(U_1) \wedge \bigwedge_{j \in J_1} \mathrm{Inf}(L_{1,j})\big) \vee \ldots \vee \big(\mathrm{Fin}(U_n) \wedge \bigwedge_{j \in J_n} \mathrm{Inf}(L_{n,j})\big)$$

For a depiction of the syntax tree see Figure 5.5. The Tseytin transformation introduces a slack variable x_k for every generalized Rabin pair k. The generalized Rabin pair $\mathrm{Fin}(U_k) \wedge \mathrm{Inf}(L_{k,1}) \wedge \ldots \wedge \mathrm{Inf}(L_{k,n_k})$ is represented by the clauses $(\neg x_k \vee \mathrm{Fin}(U_k))$, $(\neg x_k \vee \mathrm{Inf}(L_{k,1}))$ up to $(\neg x_k \vee \mathrm{Inf}(L_{k,n_k}))$. The choice for Rabin pair k is depends on the clause $\neg y_{k-1} \vee x_k \vee y_k$. The clause becomes *true* if y_{k-1} becomes *false*, if x_k becomes *true*, or if y_k becomes *true*. Variable y_{k-1} becomes false only if at least one Rabin pair with index smaller than k has been satisfied. The variable x_k becomes true only if Rabin pair k is satisfied. Alternatively, also a Rabin pair with index greater than k can be satisfied, which corresponds to variable y_k being *true*.

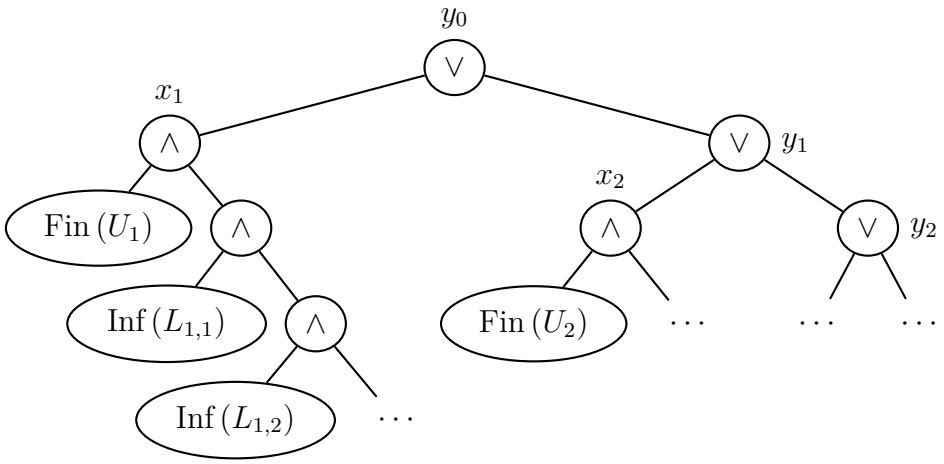


Figure 5.5: The syntax tree of a generalized Rabin condition.

The overall CNF formula for a Rabin condition is as follows:

$$\begin{aligned}\mathsf{cnf}(\Phi) =& y_0 \wedge \big(\neg y_0 \vee x_1 \vee y_1\big) \wedge \big(\neg x_1 \vee \mathrm{Fin}\,(U_1)\big) \wedge \\ & \big(\neg x_1 \vee \mathrm{Inf}\,(L_{1,1})\big) \wedge \ldots \wedge \big(\neg x_1 \vee \mathrm{Inf}\,(L_{1,n_1})\big) \wedge \big(\neg y_1 \vee x_1 \vee y_2\big) \wedge \ldots\end{aligned}$$

DPLL algorithm for generalized Rabin acceptance. For describing the behavior on $\mathsf{cnf}(\Phi)$ for a generalized Rabin condition Φ we will use the two following notations: $\mathbf{X_k}$ describes the clauses for the k-th generalized Rabin pair, i.e., $\mathbf{X_k} = (\neg x_k \vee \mathrm{Fin}\,(U_k)) \wedge \bigwedge_{j \in J_k} (\neg x_k \vee \mathrm{Inf}\,(L_{k,j}))$. $\mathbf{Y_k}$ intuitively describes the remainder of $\mathsf{cnf}(\Phi)$ if only Rabin pairs of index at least $k+1$ should be checked, i.e., $\mathbf{Y_k} = \bigwedge_{k \leq j < n-2} (\neg y_j \vee x_{j+1} \vee y_{j+1}) \wedge (\neg y_{n-2} \vee x_{n-1} \vee x_n) \wedge \bigwedge_{k<j\leq n} \mathbf{X_j}$. With Figure 5.5 it becomes clear that the DPLL-algorithm on $\mathsf{cnf}(\Phi)$ works in polynomial time: At first, the unit clause y_0 is chosen, and the assignment $y_0 \mapsto \mathit{true}$ is propagated. After that, there exists a choice between slack variables y_j and x_j each with $j > 0$. Assume the variable x_k is chosen. Setting x_k to *true* transforms the clauses $\mathbf{X_k}$ to a unit clauses by removing $\neg x_k$ in every clause. Therefore the k-th Rabin pair is checked. Additionally, variable y_k becomes pure. The exhaustive application of the pure literal rule leads to the assignment $y_j \leftarrow \mathit{false}$ for $j \geq k$, $y_j \leftarrow \mathit{true}$ for $0 < j < k$ and $x_j \leftarrow \mathit{false}$ for $j \neq k$. Setting x_j to *false* removes every clause in $\mathbf{X_j}$ and leaves a situation analogous to above.

We now assume the literal y_k is chosen. Setting y_k to *true* results in the removal of Rabin pairs with an index smaller or equal to k. In particular, by the pure literal rule every y_j with $0 < j < k$ is set to *true*, and every x_j with $0 < j \leq k$ is set to *false*. What remains of $\mathsf{cnf}(\Phi)$ are the clauses $(x_{k+1} \vee y_{k+1})$, $\mathbf{X_{k+1}}$ and $\mathbf{Y_{k+1}}$ leaving a situation similar after applying the unit rule to $\mathsf{cnf}(\Phi)$.

Dual to $y_k \leftarrow \mathit{true}$ assigning y_k to *false* causes y_{k+1} and x_{k+1} being pure, both are set to *false*, thus removing the clauses $\mathbf{X_{k+1}}$. Overall, the thorough use of the pure-literal rule removes every clause in $\mathbf{Y_k}$, thus leaving only the first up to $k-1$-th

generalized Rabin pair to be checked, which is done in polynomial time by analogous arguments.

5.3.5 Streett acceptance

Tseytin transformation for Streett acceptance. We denote a Streett acceptance as $\Phi = (\mathrm{Fin}\,(U_1) \vee \mathrm{Inf}\,(L_1)) \wedge \ldots \wedge (\mathrm{Fin}\,(U_n) \vee \mathrm{Inf}\,(L_n))$. The syntax tree is depicted in Figure 5.6. Despite a Streett condition being already in CNF, the Tseytin transformation introduces two slack variables for every Streett pair. A Streett pair is mimicked by the clause $\neg x_k \vee \mathrm{Fin}\,(U_k) \vee \mathrm{Inf}\,(L_k)$. The clause $\neg y_{k-1} \vee x_k$ signals that the k-th Streett pair should be true. The clauses $\neg y_{k-1} \vee y_k$ for every $k \in \{1, \ldots, n\}$ ensures that every Streett pair has to be satisfied.

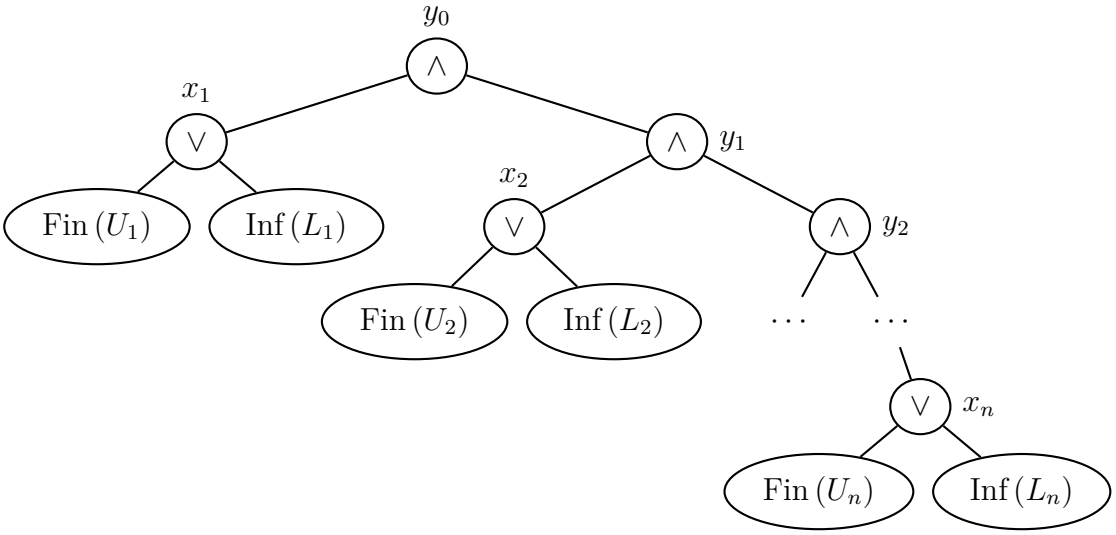


Figure 5.6: The syntax tree of a Streett condition.

Overall, the transformed formula is:

$$\begin{aligned}\mathsf{cnf}(\Phi) =& y_0 \wedge \big(\neg y_0 \vee x_1\big) \wedge \big(\neg x_1 \vee \mathrm{Fin}\,(U_1) \vee \mathrm{Inf}\,(L_1)\big) \wedge \\ & \big(\neg y_0 \vee y_1\big) \wedge \ldots \wedge \big(\neg x_n \vee \mathrm{Fin}\,(U_n) \vee \mathrm{Inf}\,(L_n)\big)\end{aligned}$$

DPLL for Streett acceptance. At first, the DPLL-algorithm reverts the Tseytin transformation by the application of the unit rule: In the k-th iteration there is a unit clause y_{k-1}. Setting y_{k-1} to *true* generates two new unit clauses, namely x_k and y_k. Again, apply the unit rule to x_k transforms the clause $\neg x_k \vee \mathrm{Fin}\,(U_k) \vee \mathrm{Inf}\,(L_k)$ to $\mathrm{Fin}\,(U_k) \vee \mathrm{Inf}\,(L_k)$, therefore generating the k-th Streett pair as a clause. Then, the $k+1$-th iteration starts with applying the unit rule to y_k. As a whole, applying the unit rule iteratively transforms $\mathsf{cnf}(\Phi)$ back to Φ.

For Streett acceptance, the unit rule suffices to ensure a polynomial-time algorithm for checking, whether an MEC contains an accepting EC: Let Δ be the transitions

of an MEC which should be checked against a Streett acceptance $\Phi = \big(\mathrm{Fin}\,(U_1) \vee \mathrm{Inf}\,(L_1)\big) \wedge \ldots \wedge \big(\mathrm{Fin}\,(U_n) \vee \mathrm{Inf}\,(L_n)\big)$. First, we intersect all sets with Δ, i.e., we obtain

$$\big(\mathrm{Fin}\,(U_1 \cap \Delta) \vee \mathrm{Inf}\,(L_1 \cap \Delta)\big) \wedge \ldots \wedge \big(\mathrm{Fin}\,(U_n \cap \Delta) \wedge \mathrm{Inf}\,(L_n \cap \Delta)\big)$$

We remove all literals of the form $\mathrm{Inf}\,(\varnothing)$ (meaning *false*) and remove all clauses with $\mathrm{Fin}\,(\varnothing)$ (with the meaning *true*). Now, if there is in every clause an $\mathrm{Inf}\,(\cdot)$ literal, the formula is fulfilled and therefore satisfiable. If not, then there exists a unit clause of the form $\mathrm{Fin}\,(F_k \cap \Delta)$. Applying the unit rule means pruning MEC $\mathcal{T}$. We obtain new MECs $\mathcal{T}_1, \ldots, \mathcal{T}_k$. Now checking whether Φ is satisfied by an EC in $\mathcal{T}$ reduces now to checking whether there exists MEC $\mathcal{T}_j \in \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$ such that it has an accepting EC for Φ'.

Complexity. The argument for polynomial time complexity follows [BGC09a]. As above, we go through every clause, and either it contains a non-empty $\mathrm{Inf}\,(\cdot)$-literal (meaning, that at the current moment, it is assumed to be *true*), or it is a unit literal of the form $\mathrm{Fin}\,(U_k)$. So, the only branching happens on the MDP side, where we prune an MEC, and divide into several new MECs and check them recursively.

Note that the recursion depth is at most $\min(|\Phi|, |\mathcal{T}|)$. Also, after pruning, we obtain at most $|\mathcal{T}| - 1$ new MECs to branch in. With a linear time for finding the new MECs, we obtain $\mathcal{O}(|\mathcal{T}|^2 \cdot \min(|\mathcal{T}|, |\Phi|))$ as overall complexity.

Emerson-Lei automata. Analogously to positivity for MDPs, for emptiness of EL automata the choice of the branching literal can influence the runtime of Algorithm 4. The presented algorithm works for a generalized Rabin and a Streett acceptance in polynomial time as well. In particular, the automata-theoretic analog of Algorithm 4 works for a Streett acceptance as described in [EL87] and for a generalized Rabin acceptance as the canonical approach.

5.4 Implementation and Experiments

Our experimental evaluation consists of two parts: At first, we evaluate our translation by comparing the automata sizes and acceptance sizes. The second contribution in our evaluation considers probabilistic model checking with the help of automata. For every experiment, we set a time limit of 30 minutes and a memory limit of 10 GB for every process.[2] The according implementations, and data logs can be found at [Mül18].

[2] All experiments were carried out on a computer with two Intel E5-2680 8-core CPUs at 2.70 GHz with 384GB of RAM running Linux.

5.4.1 Automata Sizes

For the comparison of the acceptance conditions, we rely on counting the number of $\mathrm{Fin}(\cdot)$ and $\mathrm{Inf}(\cdot)$ occurring in the acceptance condition. We compare our tool `Delag` with `Rabinizer` version 3.1 [EKS16] and `ltl2tgba` of `SPOT` version 2.5 [Dur+16]. Our benchmark consists of 94 LTL formulas from [SB00; DAC99; EH00] where for 34 formulas `Delag` was able to translate a formula completely without using an external tool. For these formulas we do not need to rely on an external tool translating LTL to deterministic automata. Should we require external tools to translate parts of the formula, as described in Section 5.1.4, we use `ltl2tgba` of `SPOT` as the fallback solution.

Overall, `Delag` produced automata with a minimal state space in 77 cases among `ltl2tgba` and `Rabinizer`, slightly surpassed by `ltl2tgba` with 78 formulas. For the comparison of the acceptance, `Delag` has delivered the smallest acceptance for 59 formulas, whereas `ltl2tgba` could produce an automaton with a minimal acceptance condition for 82 formulas. As can be seen in Table 5.1, `Delag`, `ltl2tgba` and `Rabinizer` show roughly the same behavior, generating for 36 vs. 37 vs. 35 formulas automata with size less or equal than 3, with a slight advantage for `Delag` producing more automata of size one.

Number of states $\leqslant x$	$\leqslant 1$	$\leqslant 2$	$\leqslant 3$	$\leqslant 4$	$\leqslant 6$	$\leqslant 10$	$\leqslant 20$	> 20
`Delag`	9	17	36	59	75	87	90	4
`SPOT`	6	17	37	60	78	89	92	2
`Rabinizer`	6	15	35	53	75	84	91	3

Table 5.1: Overview of the number of automata generated by the tools `Delag`, `SPOT` and `Rabinizer` with an upper bound of states.

The situation differs for the sizes of the acceptance condition (see Table 5.2): `ltl2tgba` generates 72 automata with acceptance size 1 whereas `Delag` generates 50 automata with acceptance size 1. For bigger acceptance sizes the number of generated automata are similar for `ltl2tgba` and `Delag`. In comparison, `Rabinizer` tends to produce automata with bigger acceptance sizes.

Acceptance size $\leqslant x$	$\leqslant 1$	$\leqslant 2$	$\leqslant 3$	$\leqslant 4$	$\leqslant 6$	$\leqslant 10$	> 10
`Delag`	50	79	83	83	90	91	3
`ltl2tgba`	72	84	84	86	93	94	0
`Rabinizer`	20	34	54	67	81	88	6

Table 5.2: Overview of the number of automata generated by the tools `Delag`, `ltl2tgba` and `Rabinizer` with an upper bound for the acceptance sizes.

Now, we consider a property specifically designed in such a way, that Emerson-Lei automata can reflect the structure of the corresponding LTL formula in the acceptance condition, whereas generalized Rabin (or Streett) automata has to transform the

acceptance into disjunctive (or conjunctive) normal form, resulting in a big acceptance condition. For this, we define two mutually recursive formula patterns modeling Rabin and Streett conditions:

$$\varphi_{\mathrm{R},0} = \Diamond\Box a_0 \wedge \Box\Diamond b_0 \qquad \varphi_{\mathrm{R},n+1} = (\Diamond\Box a_{n+1} \wedge \Box\Diamond b_{n+1}) \vee \varphi_{\mathrm{S},n}$$
$$\varphi_{\mathrm{S},0} = \Diamond\Box a_0 \vee \Box\Diamond b_0 \qquad \varphi_{\mathrm{S},n+1} = (\Diamond\Box a_{n+1} \vee \Box\Diamond b_{n+1}) \wedge \varphi_{\mathrm{R},n}$$

It is clear from Definition 5.7 that the presented translation in Section 5.1 uses $2n + 2$ acceptance sets for $\varphi_{\mathrm{R},n}$ and one state.

For the formulas $\varphi_{\mathrm{R},n}$ the results are as expected (see Table 5.3). `Delag` produces always the smallest acceptance with a one-state automaton, whereas the acceptance sizes of the automata produced by `Rabinizer` grow faster, e.g., for $n = 5$ and $n = 7$ `Rabinizer` produces an automaton with acceptance size 45 and 109, respectively. Both `Delag` and `Rabinizer` produce one state automata. `ltl2tgba` behaves differently: The state space size of the automata grows with n: for $n = 1$ `ltl2tgba` produces an automaton with 7 states and an acceptance size of 4, whereas for $n = 3$ the state space increased to 21889 states and an acceptance size of 20. For $n > 3$ we were not able to produce automata with `ltl2tgba`.

$n =$	0	1	2	3	4	5	6	7
`Delag`	2	4	6	8	10	12	14	16
`ltl2tgba`	2	4	8	20	–	–	–	–
`Rabinizer`	2	5	7	17	19	45	47	109

Table 5.3: Acceptance sizes for the alternating formula $\varphi_{\mathsf{R},n}$; – means timeout or memout.

For the evaluation of the history, we took the formula pattern $\varphi_{\mathcal{H},n}$:

$$\varphi_{\mathcal{H},n} = \begin{cases} \big(\Diamond\Box(a \vee \bigcirc^n b)\big) \vee \varphi_{\mathcal{H},n-1} & \text{if } n \text{ is even} \\ \big(\Diamond\Box(\neg a \vee \bigcirc^n b)\big) \vee \varphi_{\mathcal{H},n-1} & \text{otherwise} \end{cases}$$

Every subformula $a \vee \bigcirc^n b$ (or $\neg a \vee \bigcirc^n b$) commits the first position or the n-th position. So only two out of n positions may be fixed, and hence we can share a lot of the state space between the $\Diamond\Box$ formulas.

The results can be found in Table 5.4. The state space of `ltl2tgba` grows faster than `Delag`, the former being only capable to produce automata up to $n = 5$ before hitting the memory limit. For `Rabinizer`, we were not able to produce automata for $n \geqslant 4$, since `Rabinizer` supports only a limited number of acceptance set. This shows, that the acceptance condition grows immensely.

5.4.2 Implementations and Experiments in `PRISM`

We have implemented a routine for the analysis of MDPs in `PRISM` version 4.3.1 dev. Here we compare the behavior of `PRISM` if the four tools `Delag`, `ltl2tgba` from

	$n =$	0	1	2	3	4	5	6	7
`Delag`	#States	1	2	4	8	16	32	64	128
	Acc. size	1	2	3	4	5	6	7	8
`ltl2tgba`	#States	2	4	21	170	1816	22196	–	–
	Acc. size	2	2	2	2	2	2	–	–
`Rabinizer`	#States	1	2	5	11	–	–	–	–
	Acc. size	1	3	7	19	–	–	–	–

Table 5.4: Automata sizes and number of acceptance sets for $\varphi_{\mathcal{H},n}$; – means timeout or memout.

`SPOT`, `Rabinizer` and the `PRISM` reimplementation of `ltl2dstar` are employed as automata generation tools. As case study we consider the IEEE 802.11 Wireless LAN Handshaking protocol [KNS02] from the `PRISM` benchmark suite [KNP12]. It describes a resolving mechanism to stop interference if two stations want to send a message at the same time. The key trick is that all participating stations listen to interference, and if a message has become garbled, the stations waits a random amount of time (limited by an upper bound called `Backoff`) and then tries to resend the message. We used the following properties:

- $\varphi_1 = \bigwedge_{1 \le i \le n} \square\,(\texttt{garbled}_i \rightarrow \lozenge\,\texttt{correct}_i)$
  "If a message from sender i has been garbled, it will be sent correctly in the future"

- $\varphi_2 = \bigwedge_{1 \le i \le n} \lozenge\,\texttt{correct}_i$: "Every sender sends at least one message correctly."

- $\varphi_3 = \bigwedge_{1 \le i \le n} \texttt{wait}_i\ \mathcal{U}\ (\texttt{wait}_i \wedge \square^{\le k}\,\texttt{free})$
  where $\square^{\le k}\texttt{free} = \texttt{free} \wedge \bigcirc\,\texttt{free} \wedge \ldots \wedge \bigcirc^k \texttt{free}$
  "The first time every station wants to send, the channel remains free for k steps"

- $\varphi_4 = \bigwedge_{1 \le i \le n} (\square\lozenge\,\texttt{wait}_i) \rightarrow (\square\lozenge\,\texttt{correct}_i)$
  "Every station, that wants to send a message infinitely often, is able to send a message correctly infinitely often "

- $\varphi_5 = \left(\bigwedge_{1 \le i \le n} \lozenge\,\texttt{correct}_i\right) \wedge \left(\bigwedge_{1 \le i \le n} (\square\lozenge\,\texttt{wait}_i) \rightarrow (\square\lozenge\,\texttt{correct}_i)\right)$
  "Every station satisfies both the reachability formula φ_2 and the fairness formula φ_4"

Every property can be translated directly by `Delag` without external tools, except φ_1, for which we translate the subformulas $\square\,(\texttt{garbled}_i \rightarrow \lozenge\,\texttt{correct}_i)$ with `ltl2tgba` and then build the product. So φ_1 should be seen as a benchmark for the product construction.

For all properties we asked for the minimal ($\Pr^{\mathsf{min}}(\cdot)$) or maximal ($\Pr^{\mathsf{max}}(\cdot)$) probability of the IEEE 802.11 handshaking model with two stations and a `Backoff` of at most 3 to satisfy the property. If a formula has a window length (e.g. $\Box^{\leq k}$) we uniformly choose $k = 6$. Table 5.5 lists some measured time values and automata/product sizes. In contrast to the evaluation of the good-for-games approach in Section 3.3, all `PRISM` experiments in this section were carried out with the `hybrid` engine, an engine that combines symbolic and explicit data structures offering a good compromise.

Property	PRISM Delag			PRISM ltl2tgba			PRISM Rabinizer			PRISM std		
	$\lvert\mathcal{A}\rvert$	BDD size $\mathcal{M}\otimes\mathcal{A}$	t_{MC}	$\lvert\mathcal{A}\rvert$	BDD size $\mathcal{M}\otimes\mathcal{A}$	t_{MC}	$\lvert\mathcal{A}\rvert$	BDD size $\mathcal{M}\otimes\mathcal{A}$	t_{MC}	$\lvert\mathcal{A}\rvert$	BDD size $\mathcal{M}\otimes\mathcal{A}$	t_{MC}
$\Pr^{\mathsf{min}}(\varphi_1)$	4	31,861	8.7 s	5	44,172	9.8 s	4	31,861	32.7 s	15	45,095	122.5 s
$\Pr^{\mathsf{min}}(\varphi_2)$	4	61,711	188.7 s	4	61,719	196.8 s	4	61,719	189.2 s	5	61,759	167.7 s
$\Pr^{\mathsf{min}}(\varphi_3)$	20	46,013	27.2 s	20	46,109	27.6 s	72	47,114	28.3 s	21	46,025	15.7 s
$\Pr^{\mathsf{min}}(\varphi_4)$	1	30,091	50.6 s	5	30,473	8.7 s	1	30,091	44.8 s	49	39,141	62.5 s
$\Pr^{\mathsf{max}}(\varphi_4)$	1	30,091	7.9 s	35	153,906	600.1 s	1	30,091	5.9 s	–	–	–
$\Pr^{\mathsf{min}}(\varphi_5)$	4	61,711	130.0 s	21	65,469	96.0 s	4	61,719	127.5 s	197	80,632	161.5 s
$\Pr^{\mathsf{max}}(\varphi_5)$	4	61,711	162.7 s	38	176,628	1152.7 s	4	61,719	164.0 s	–	–	–

Table 5.5: PRISM runtimes (t_{MC}) for the IEEE 802.11 case study enhanced with automata sizes ($|A|$) and the number of BDD nodes in the product ($|\mathcal{M}\otimes\mathcal{A}|$). − stands for timeout.

First, the generation time for every automaton was below 1.0 s, except for `Rabinizer` at $\Pr^{\min}(\varphi_3)$ where it was 1.8 s and for the `ltl2dstar` reimplementation in `PRISM` at $\Pr^{\max}(\varphi_4)$ (3.8 s) as well as at $\Pr^{\max}(\varphi_5)$ (14.1 s). In two cases `PRISM Delag` was the fastest. For $\Pr^{\min}(\varphi_4)$ `PRISM ltl2tgba` took only 8.7 s in comparison to 50.6 s for `PRISM Delag` despite the smaller automaton, as one heuristic applies for `PRISM ltl2tgba` did not apply for `PRISM Delag`: For the analysis of the MECs we always check at first, whether the whole MEC satisfies the acceptance condition, and only if this is not the case, we look for accepting sub-end-components within the MEC. For `PRISM ltl2tgba` the whole MEC was accepting, but for `PRISM Delag` one has to search for an accepting sub-end-component. Since in a symbolic representation SCC enumeration is costly, `PRISM ltl2tgba` was much faster.

In general, one can see, that `Delag` always produced the smallest automaton resulting in the smallest number of BDD nodes in the product and comparatively small model checking times for `PRISM Delag`.

5.5 Conclusion

We presented a general framework based on the product construction and specialized translations for fragments of LTL to build deterministic Emerson-Lei automata. In particular, for the important fairness fragment we established an efficient construction, where the state space only depends on the nesting depth of $\bigcirc$, and the acceptance condition reflects the complexity of the formula. The general construction applies a range of additional optimizations, such as pushing temporal operators down the syntax tree, piggybacking to reduce the number of acceptance sets and sharing of equal automata parts. In particular, our history buffer approach reduces the state space, since the buffer can be shared between automata for different subformulas. If a formula does not belong to one of our explicitly supported fragments, we can run an external LTL-to-deterministic-automaton translator and incorporate the resulting automaton via product construction and lifting.

Benchmarking this approach has shown the potential of our method. Standard benchmarks highlight the potential of allowing more complex acceptance conditions, our tool produced automata with state numbers comparable to `SPOT`.

However, the heuristics presented here are not complete, and this approach should be understood as a framework. So, one direction for future work is to add more explicitly supported LTL fragments. Another point would be to analyze the subformulas which cannot be translated directly and choose an external tool that behaves well for these specific subformulas. For example, it is well-known, that obligation LTL formulas can be translated to weak DBA, and then efficiently minimized. This is implemented in `SPOT`. Another direction one could take is a deeper look into is starting with a non-deterministic Büchi automaton and trying to find small deterministic automata with complex acceptance conditions. Of course, general methods to shrink the state space like bisimulation could be also applied. Also, the particular ingredients of our transformation could be optimized further, e.g., the history could

be allocated dynamically, and therefore reduce the state space even further without increasing the acceptance condition complexity.

Of course, both approaches could be combined, and intermediate steps over non-deterministic automata are only done, if the LTL formula cannot be dealt with by `Delag` directly.

For MDP analysis, the Emerson-Lei acceptance adds some difficulty. In fact, if deciding satisfiability for a class of Boolean formulas is NP-complete, then deciding positiveness for MDPs and an Emerson-Lei acceptance of the corresponding Emerson-Lei acceptance class is NP-complete as well. Still, for subclasses like acceptances without any $\mathrm{Fin}(\cdot)$ variable, polynomial time checks are possible. For the general case the DPLL algorithm for solving the satisfiability of Boolean formula can be adapted to the Emerson-Lei setting. To achieve the required conjunctive normal form, we refined the Tseytin transformation. This transformation introduces slack variables which are treated by the consecutive DPLL-algorithm. The slack variables provide an interesting heuristic for the choice of the branch variable in our DPLL algorithm: We always prefer slack variables over $\mathrm{Fin}(\cdot)$ variables. This heuristic allows a polynomial time behavior for Rabin and Streett acceptances similar to already known approaches.

The benchmarks of our approach in the area of PMC show that our DPLL algorithm causes only a very small overhead, but benefits from the small state space. Thus, the time for the overall model checking process were for 5 out of 7 cases in the same order of magnitude as the best time of `PRISM Rabinizer` and `PRISM ltl2tgba`.

6 Conclusion

In this thesis we presented three new automata-based approaches different from the standard approach that employs deterministic Rabin automata in order to avoid or cope with the double-exponential blow-up.

Good-for-games automata were a promising candidate for MDP analysis. We showed that the GFG strategy of the good-for-games automaton and the maximal scheduler of an MDP share some similarities. The GFG strategy tries to construct an accepting run for an accepted word depending on the history, whereas the maximizing scheduler tries to maximize the probability depending on the history. Thus, a modified product construction and a standard MEC and reachability analysis are sufficient for MDP analysis against a GFG automaton specification. The overall approach starting from LTL can be done in double-exponential time, thus meeting the lower bound for MDP analysis under LTL specifications. Nonetheless, practical evaluation showed that good-for-games automata fell behind the standard approach of using [GO01] to generate a non-deterministic Büchi automaton and Safra's determinization [Saf88] to determinize it.

Unambiguous automata on the other hand showed an advantage over deterministic automata both on a theoretical level and in a practical evaluation. The single-exponential blow-up of the LTL-to-UBA translation gives UBA an edge over deterministic automata, but the actual model checking process becomes more complicated although still polynomial. Still, the performance of the overall process heavily depends on efficient LTL-to-UBA translators.

In 2017, Boker, Kupferman and Skrzypczak [BKS17] have proven that every unambiguous GFG automaton can be transformed to an equivalent deterministic automaton by removing unnecessary transitions and states. Thus, the combination of unambiguity and good-for-games is not a promising candidate for performance improvements.

The Emerson-Lei acceptance differs from unambiguity and good-for-games, as it works on the acceptance level, not on the non-determinism of the automaton. The product construction allows a reflection of the Boolean structure of the input LTL formula into the acceptance condition, not into the state space of the automaton. Since every Emerson-Lei automaton is actually a Muller automaton with a symbolically represented acceptance condition, the double-exponential blow-up from LTL to deterministic automata cannot be avoided by the Emerson-Lei acceptance. Still, in the automata generation part our tool `Delag` could compete with other standard automata tools like `Rabinizer` and `ltl2tgba` from `SPOT`.

In the application for model checking, EL automata are well-suited for the analysis of Markov chains, as checking $\Pr_{\mathcal{M}}(\Phi) > 0$ is still possible in polynomial time. For

Markov decision processes, the situation becomes more complicated. The positivity problem relates closely to satisfiability of Boolean formulas, and thus, becomes NP-complete. As solution for checking $\Pr^{\max}_{\mathcal{M}}(\Phi) > 0$ we took inspiration from the DPLL algorithm. As an advantage, our DPLL-based algorithm works in polynomial time for the commonly used acceptances generalized Rabin and Streett. As our practical evaluation on the WLAN handshaking protocol shows, it is worth to take the comparably small overhead of solving the NP-complete positivity problem.

Future Work. The bad performance of the good-for-games approach in the benchmarks comes from the GFG automata generation algorithm developed by Henzinger and Piterman. This observation asks for an improved generation algorithm. Very recently, [KM18] suggested a modified version of Safra's determinization algorithm for translating an NBA to a good-for-games automaton. To the best of our knowledge, there is no implementation publicly available and therefore an evaluation of their proposal is an open task. If the generation algorithm performs well, the usage of good-for-games automata for probabilistic model checking could turn out more positive, and if the automaton suits an explicit representation, renders GFG-based MDP analysis with explicit representation feasible.

The principle that smaller automata likely lead to a more efficient model checking process also holds for our UBA-based approach to Markov chain analysis. The problem of UBA generation has not been studied in depth, but is current research by us. Currently, only one LTL-to-UBA translator is available, which uses an adaption of [Cou99], see also [Dur17]. A comparison in a symbolic setting as well as a comparison with the other single-exponential approaches of [BRV04] and [CSS03] is also still open. Apart from applications in probabilistic model checking, a very intriguing open problem is universality checking ("Is every infinite word accepted?") for UBA. We can only give a PSPACE upper bound as this problem can be solved in PSPACE for NBA.

Obviously, the single-exponential generation of UBA from LTL forbids a direct usage for the MDP analysis, as MDP analysis under LTL is 2EXPTIME-complete. However, unambiguous Büchi automata might be a good starting point for deterministic-in-the-limit automata to achieve quantitative MDP analysis similar to [Sic+16].

A natural idea would be to combine unambiguity with Emerson-Lei acceptance. Still, one would have to rewrite the LTL formula on the top-level according to [Dur17], and then apply a product construction for conjunctions and a union of automata for disjunctions. Nevertheless, as every known translation which produces unambiguous, but not deterministic automata, aims at (generalized) Büchi acceptance, the resulting overall acceptance would be without any $\mathrm{Fin}(\cdot)$ atom.

As we have identified two LTL fragments with a natural translation to deterministic Emerson-Lei automata, the translation of full LTL remains an open question. A reasonable starting point would be NBA and then a modification of a suitable determinization algorithm.

For the positivity problem we employed only a rather simple DPLL-based algorithm,

known techniques like conflict-driven clause learning could improve the performance even further. Not only improving the efficiency of the actual model checking algorithm would lead to a better overall performance, but also generating smaller deterministic Emerson-Lei automata. However, minimization of Emerson-Lei (or Muller) automata is an unsolved problem, but succinctness results in comparison to Rabin and Streett automata [Bok17a] demonstrate the potential of Emerson-Lei automata.

Bibliography

[AD94] Rajeev Alur and David L. Dill. "A Theory of Timed Automata". In: *Theoretical Computer Science* 126.2 (1994), pp. 183–235.

[AL04] Rajeev Alur and Salvatore La Torre. "Deterministic Generators and Games for LTL Fragments". In: *ACM Transactions on Programming Languages and Systems* 5.1 (2004), pp. 1–25.

[Alf97] Luca de Alfaro. "Formal Verification of Probabilistic Systems". PhD thesis. Stanford University, Department of Computer Science, 1997.

[Arn85] André Arnold. "Deterministic and non ambiguous rational omega-languages". In: *Automata on Infinite Words*. Vol. 192. Lecture Notes in Computer Science. Springer, 1985, pp. 18–27.

[Bab+12] Tomás Babiak, Mojmír Křetínský, Vojtěch Řehák, and Jan Strejček. "LTL to Büchi Automata Translation: Fast and More Deterministic". In: *18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 7214. Lecture Notes in Computer Science. Springer, 2012, pp. 95–109.

[Bab+13a] Tomás Babiak, Thomas Badie, Alexandre Duret-Lutz, Mojmír Kretínský, and Jan Strejcek. "Compositional Approach to Suspension and Other Improvements to LTL Translation". In: *25th International Symposium on Model Checking of Software (SPIN)*. Vol. 7976. Lecture Notes in Computer Science. Springer, 2013, pp. 81–98.

[Bab+13b] Tomáš Babiak, Frantisek Blahoudek, Mojmír Křetínský, and Jan Strejcek. "Effective Translation of LTL to Deterministic Rabin Automata: Beyond the (F, G)-Fragment". In: *11th International Symposium on Automated Technology for Verification and Analysis (ATVA)*. Vol. 8172. Lecture Notes in Computer Science. Springer, 2013, pp. 24–39.

[Bab+15] Tomáš Babiak, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, Jan Křetínský, David Müller, David Parker, and Jan Strejček. "The Hanoi Omega-Automata Format". In: *27th International Conference on Computer Aided Verification (CAV)*. Vol. 9206. Lecture Notes in Computer Science. Springer, 2015, pp. 479–486.

[Bah+93] R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. "Algebraic Decision Diagrams and their Applications". In: *11th International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1993, pp. 188–191.

[Bai+16] Christel Baier, Stefan Kiefer, Joachim Klein, Sascha Klüppelholz, David Müller, and James Worrell. "Markov Chains and Unambiguous Büchi Automata". In: *28th International Conference on Computer Aided Verification (CAV) - Part I*. Vol. 9779. Lecture Notes in Computer Science. Springer, 2016, pp. 23–42.

[Bai+17] Christel Baier, Joachim Klein, Linda Leuschner, David Parker, and Sascha Wunderlich. "Ensuring the Reliability of Your Model Checker: Interval Iteration for Markov Decision Processes". In: *29th International Conference on Computer Aided Verification (CAV)*. Vol. 10426. Lecture Notes in Computer Science. Springer, 2017, pp. 160–180.

[BC04] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development*. Springer, 2004.

[BD15] Souheib Baarir and Alexandre Duret-Lutz. "SAT-Based Minimization of Deterministic ω-Automata". In: *20th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*. Vol. 9450. Lecture Notes in Computer Science. Springer, 2015, pp. 79–87.

[BGC09a] Christel Baier, Marcus Größer, and Frank Ciesinski. "Model Checking Linear-Time Properties of Probabilistic Systems". In: *Handbook of Weighted Automata*. Monographs in Theoretical Computer Science. Springer, 2009. Chap. 13, pp. 519–570.

[BGC09b] Christel Baier, Marcus Größer, and Frank Ciesinski. "Quantitative Analysis under Fairness Constraints". In: *7th International Symposium on Automated Technology for Verification and Analysis (ATVA)*. Vol. 5799. Lecture Notes in Computer Science. Springer, 2009, pp. 135–150.

[BGS00] Roderick Bloem, Harold N. Gabow, and Fabio Somenzi. "An Algorithm for Strongly Connected Component Analysis in $n \log n$ Symbolic Steps". In: *Third International Conference on Formal Methods in Computer Aided Design (FMCAD)*. Lecture Notes in Computer Science 2517. Springer-Verlag, 2000, pp. 37–54.

[Bie+03] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. "Bounded Model Checking". In: *Advances in Computers* 58 (2003), pp. 117–148.

[Bie+99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. "Symbolic Model Checking without BDDs". In: *Fifth International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*. Vol. 1579. Lecture Notes in Computer Science. Springer, 1999, pp. 193–207.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

[BKS17] Udi Boker, Orna Kupferman, and Michał Skrzypczak. "How Deterministic are Good-For-Games Automata?" In: *37th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Vol. 93. Leibniz International Proceedings in Informatics. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 18:1–18:14.

[BL10] Nicolas Bousquet and Christof Löding. "Equivalence and Inclusion Problem for Strongly Unambiguous Büchi Automata". In: *4th International Conference on Language and Automata Theory and Applications (LATA)*. Vol. 6031. Lecture Notes in Computer Science. Springer, 2010, pp. 118–129.

[Bla+17] Frantisek Blahoudek, Alexandre Duret-Lutz, Mikulás Klokocka, Mojmír Kretínský, and Jan Strejcek. "Seminator: A Tool for Semi-Determinization of Omega-Automata". In: *21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*. Vol. 46. EPiC Series in Computing. EasyChair Publications, 2017, pp. 356–367.

[BLW13a] Michael Benedikt, Rastislav Lenhardt, and James Worrell. "LTL Model Checking of Interval Markov Chains". In: *19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 7795. Lecture Notes in Computer Science. Springer, 2013, pp. 32–46.

[BLW13b] Michael Benedikt, Rastislav Lenhardt, and James Worrell. "Two Variable vs. Linear Temporal Logic in Model Checking and Games". In: *Logical Methods in Computer Science* 9.2 (2013).

[BLW14] Michael Benedikt, Rastislav Lenhardt, and James Worrell. *Model Checking Markov Chains Against Unambiguous Büchi Automata*. arXiv:1405.4560. 2014.

[Bok+13] Udi Boker, Denis Kuperberg, Orna Kupferman, and Michał Skrzypczak. "Nondeterminism in the Presence of a Diverse or Unknown Future". In: *40th International Conference on Automata, Languages, and Programming Colloquium (ICALP, Track B)*. Vol. 7966. Lecture Notes in Computer Science. Springer, 2013, pp. 89–100.

[Bok17a] Udi Boker. "On the (In)Succinctness of Muller Automata". In: *26th Conference on Computer Science Logic (CSL)*. Vol. 82. Leibniz International Proceedings in Informatics. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 12:1–12:16.

[Bok17b] Udi Boker. "Rabin vs. Streett Automata". In: *37th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Vol. 93. Leibniz International Proceedings in Informatics. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 17:1–17:15.

[BP79] Abraham Berman and Robert J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. Academic Press, 1979.

[BRV04] Doron Bustan, Sasha Rubin, and Moshe Y. Vardi. "Verifying ω-Regular Properties of Markov Chains". In: *16th International Conference on Computer Aided Verification (CAV)*. Vol. 3114. Lecture Notes in Computer Science. Springer, 2004, pp. 189–201.

[Bry86] Randal E. Bryant. "Graph-Based Algorithms for Boolean Function Manipulation". In: *IEEE Transactions on Computers* 35.8 (1986), pp. 677–691.

[Büc62] Julius Richard Büchi. "On a Decision Method in restricted Second Order Arithmetic". In: *First International Congress on Logic, Methodology and Philosophy of Science (CLMPS)*. Stanford University Press, 1962, pp. 1–11.

[Bur+92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. "Symbolic Model Checking: 10^20 States and Beyond". In: *Information and Computation* 98.2 (1992), pp. 142–170.

[BY04] Johan Bengtsson and Wang Yi. "Timed Automata: Semantics, Algorithms and Tools". In: *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*. Vol. 3098. Lecture Notes in Computer Science. Springer, 2004, pp. 87–124.

[CE81] Edmund M. Clarke and Ernest A. Emerson. "Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic". In: *Logics of Programs*. Vol. 131. Lecture Notes in Computer Science. Springer, 1981, pp. 52–71.

[CES86] Edmund M. Clarke, Ernest A. Emerson, and A. Prasad Sistla. "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications". In: *ACM Transactions on Programming Languages and Systems* 8.2 (1986), pp. 244–263.

[CFZ93] Edmund M. Clarke, Masahiro Fujita, and Xudong Zhao. "Multi-Terminal Binary Decision Diagrams and Hybrid Decision Diagrams". In: *Third International Workshop on Logic & Synthesis (IWLS)*. 1993, pp. 1–15.

[CGH97] Edmund M. Clarke, Orna Grumberg, and Kiyoharu Hamaguchi. "Another Look at LTL Model Checking". In: *Formal Methods in System Design* 10.1 (1997), pp. 47–71.

[CGK13] Krishnendu Chatterjee, Andreas Gaiser, and Jan Křetínský. "Automata with Generalized Rabin Pairs for Probabilistic Model Checking and LTL Synthesis". In: *25th International Conference on Computer Aided Verification (CAV)*. Lecture Notes in Computer Science. Springer, 2013, pp. 559–575.

[Cim+02] Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. "NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking". In: *14th International Conference on Computer-Aided Verification (CAV)*. Vol. 2404. Lecture Notes in Computer Science. Springer, 2002.

[CK14] Souymodip Chakraborty and Joost-Pieter Katoen. "Parametric LTL on Markov Chains". In: *8th IFIP International Conference on Theoretical Computer Science (TCS)*. Vol. 8705. Lecture Notes in Computer Science. Springer, 2014, pp. 207–221.

[CM03] Olivier Carton and Max Michel. "Unambiguous Büchi automata". In: *Theoretical Computer Science* 297.1–3 (2003), pp. 37–81.

[Col09] Thomas Colcombet. "The Theory of Stabilisation Monoids and Regular Cost Functions". In: *36 International Colloquium on Automata, Languages and Programming (ICALP)*. Vol. 5556. Lecture Notes in Computer Science. Springer, 2009, pp. 139–150.

[Col12] Thomas Colcombet. "Forms of Determinism for Automata". In: *29th International Symposium on Theoretical Aspects of Computer Science, (STACS)*. Vol. 14. Leibniz International Proceedings in Informatics. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012, pp. 1–23.

[Col15] Thomas Colcombet. "Unambiguity in Automata Theory". In: *17th International Workshop on Descriptional Complexity of Formal Systems (DCFS)*. Vol. 9118. Lecture Notes in Computer Science. Springer, 2015, pp. 3–18.

[Cou99] Jean-Michel Couvreur. "On-the-Fly Verification of Linear Temporal Logic". In: *1st World Congress on Formal Methods in the Development of Computing Systems (FM)*. Vol. 1708. Lecture Notes in Computer Science. Springer, 1999, pp. 253–271.

[ČP03] Ivana Černá and Radek Pelánek. "Distributed Explicit Fair Cycle Detection (Set Based Approach)". In: *10th International SPIN Workshop on Model Checking of Software (SPIN)*. Lecture Notes in Computer Science. Springer, 2003, pp. 49–73.

[CSS03] Jean-Michel Couvreur, Nasser Saheb, and Grégoire Sutre. "An Optimal Automata Approach to LTL Model Checking of Probabilistic Systems". In: *10th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*. Vol. 2850. Lecture Notes in Computer Science. Springer, 2003, pp. 361–375.

[CY88] Costas Courcoubetis and Mihalis Yannakakis. "Verifying temporal properties of finite-state probabilistic programs". In: *29th Symposium on Foundations of Computer Science (FOCS)*. IEEE computer society, 1988, pp. 338–345.

[CY95] Costas Courcoubetis and Mihalis Yannakakis. "The Complexity of Probabilistic Verification". In: *Journal of the ACM* 42.4 (1995), pp. 857–907.

[DAC99] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. "Patterns in Property Specifications for Finite-State Verification". In: *International Conference on Software Engineering (ICSE)*. ACM, 1999, pp. 411–420.

[Deh+17] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. "A Storm is Coming: A Modern Probabilistic Model Checker". In: *29th International Conference on Computer Aided Verification (CAV)*. Vol. 10427. Lecture Notes in Computer Science. Springer, 2017, pp. 592–600.

[Die17] Reinhard Diestel. Vol. 173. Graduate Texts in Mathematics. Springer, 2017.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. "A Machine Program for Theorem-Proving". In: *Communications of the ACM* 5.7 (1962), pp. 394–397.

[DP60] Martin Davis and Hilary Putnam. "A Computing Procedure for Quantification Theory". In: *Journal of the ACM* 7.3 (1960), pp. 201–215.

[DPC09] Alexandre Duret-Lutz, Denis Poitrenaud, and Jean-Michel Couvreur. "On-the-fly Emptiness Check of Transition-based Streett Automata". In: *7th International Symposium on Automated Technology for Verification and Analysis (ATVA'09)*. Vol. 5799. Lecture Notes in Computer Science. Springer, 2009, pp. 213–227.

[Dur+16] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. "Spot 2.0 - A Framework for LTL and ω-Automata Manipulation". In: *14th Internation Symposium on Automated Technology for Verification and Analysis (ATVA)*. Vol. 9938. Lecture Notes in Computer Science. 2016, pp. 122–129.

[Dur13] Alexandre Duret-Lutz. "Manipulating LTL Formulas Using Spot 1.0". In: *11th International Symposium on Automated Technology for Verification and Analysis (ATVA)*. Vol. 8172. Lecture Notes in Computer Science. Springer, 2013, pp. 442–445.

[Dur14] Alexandre Duret-Lutz. "LTL translation improvements in Spot 1.0". In: *International Journal of Critical Computer-Based Systems* 5.1/2 (2014), pp. 31–54.

[Dur17] Alexandre Duret-Lutz. "Contributions to LTL and ω-Automata for Model Checking". Habilitation Thesis. Université Pierre et Marie Curie (Paris 6), 2017.

[EC80] Ernest A. Emerson and Edmund M. Clarke. "Characterizing Correctness Properties of Parallel Programs Using Fixpoints". In: *7th International Conference on Automata, Languages and Programming (ICALP)*. Vol. 85. Lecture Notes in Computer Science. Springer, 1980, pp. 169–181.

[EH00] Kousha Etessami and Gerard J. Holzmann. "Optimizing Büchi Automata". In: *11th International Conference on Concurrency Theory (CONCUR)*. Vol. 1877. Lecture Notes in Computer Science. Springer, 2000, pp. 153–167.

[Ehl10] Rüdiger Ehlers. "Minimising Deterministic Büchi Automata Precisely Using SAT Solving". In: *13th International Conference on Theory and Applications of Satisfiability Testing (SAT)*. Vol. 6175. Lecture Notes in Computer Science. Springer, 2010, pp. 326–332.

[EK14] Javier Esparza and Jan Křetínský. "From LTL to Deterministic Automata: a Safraless Compositional Approach". In: *26th International Conference on Computer Aided Verification (CAV)*. Vol. 8559. Lecture Notes in Computer Science. Springer, 2014, pp. 192–208.

[EKS16] Javier Esparza, Jan Křetínský, and Salomon Sickert. "From LTL to Deterministic Automata - A Safraless Compositional Approach". In: *Formal Methods in System Design* 49.3 (2016), pp. 219–271.

[EKS18] Javier Esparza, Jan Křetínský, and Salomon Sickert. "One Theorem to Rule Them All: A Unified Translation of LTL into ω-Automata". In: *33th ACM/IEEE Symposium on Logic in Computer Science (LICS)*. to appear. 2018.

[EL85] Ernest A. Emerson and Chin-Laung Lei. "Modalities for model checking: branching time strikes back". In: *12th ACM Symposium on Principles of Programming Languages (POPL)*. ACM, 1985, pp. 84–96.

[EL87] Ernest A. Emerson and Chin-Laung Lei. "Modalities for Model Checking: Branching Time Logic Strikes Back". In: *Science of Computer Programming* 8.3 (1987), pp. 275–306.

[FL15] Dana Fisman and Yoad Lustig. "A Modular Approach for Büchi Determinization". In: *26th International Conference on Concurrency Theory (CONCUR)*. Vol. 42. Leibniz International Proceedings in Informatics. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 368–382.

[Fog+13] Seth Fogarty, Orna Kupferman, Moshe Y. Vardi, and Thomas Wilke. "Profile Trees for Büchi Word Automata, with Application to Determinization". In: *Fourth International Symposium on Games, Automata, Logics and Formal Verification (GandALF)*. Vol. 119. Electronic Proceedings of Theoretical Computer Science. 2013, pp. 107–121.

[FV96] Jerzy Filar and Koos Vrieze. *Competitive Markov Decision Processes*. Springer, 1996.

[Ger+95] Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper. "Simple On-the-fly Automatic Verification of Linear Temporal Logic". In: *15th IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification (PSTV)*. Vol. 38. IFIP Conference Proceedings. Chapman & Hall, 1995, pp. 3–18.

[GO01] Paul Gastin and Denis Oddoux. "Fast LTL to Büchi Automata Translation". In: *13th International Conference on Computer Aided Verification, (CAV)*. Vol. 2102. Lecture Notes in Computer Science. Springer, 2001, pp. 53–65.

[GTW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, eds. *Automata, Logics, and Infinite Games: A Guide to Current Research*. Vol. 2500. Lecture Notes in Computer Science. Springer, 2002.

[Hah+14] Ernst Moritz Hahn, Yi Li, Sven Schewe, Andrea Turrini, and Lijun Zhang. "ISCASMC: A Web-Based Probabilistic Model Checker". In: *19th International Symposium on Formal Methods (FM)*. Vol. 8442. Lecture Notes in Computer Science. Springer, 2014, pp. 312–317.

[Hah+15] Ernst Moritz Hahn, Guangyuan Li, Sven Schewe, Andrea Turrini, and Lijun Zhang. "Lazy Probabilistic Model Checking without Determinisation". In: *26th International Conference on Concurrency Theory (CONCUR)*. Vol. 42. Leibniz International Proceedings in Informatics. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 354–367.

[HJ13] Roger A. Horn and Charles R. Johnson. *Matrix analysis*. 2nd. Cambridge University Press, 2013.

[HJ94] Hans Hansson and Bengt Jonsson. "A Logic for Reasoning about Time and Reliability". In: *Formal Aspects of Computing* 6 (1994), pp. 512–535.

[Hol04] Gerard J. Holzmann. *The SPIN Model Checker - Primer and Reference Manual*. Addison-Wesley, 2004. ISBN: 978-0-321-22862-8.

[Hos04] Wolfgang Hoschek. *The Colt Distribution: Open Source Libraries for High Performance Scientific and Technical Computing in Java*. 2004.

[HP06] Thomas A. Henzinger and Nir Piterman. "Solving Games Without Determinization". In: *20th International Workshop on Computer Science Logic (CSL)*. Vol. 4207. Lecture Notes in Computer Science. Springer, 2006, pp. 395–410.

[HSP83] Sergiu Hart, Micha Sharir, and Amir Pnueli. "Termination of Probabilistic Concurrent Program". In: *Transactions on Programming Languages and Systems* 5.3 (1983), pp. 356–380.

[HSS17] Yo-Sub Han, Arto Salomaa, and Kai Salomaa. "Ambiguity, Nondeterminism and State Complexity of Finite Automata". In: *Acta Cybernetica* 23.1 (2017), pp. 141–157.

[IL12] Dimitri Isaak and Christof Löding. "Efficient Inclusion Testing for Simple Classes of Unambiguous ω-Automata". In: *Information Processing Letters* 112.14-15 (2012), pp. 578–582.

[Kat+11] Joost-Pieter Katoen, Ivan S. Zapreev, Ernst Moritz Hahn, Holger Hermanns, and David N. Jansen. "The Ins and Outs of The Probabilistic Model Checker MRMC". In: *Performance Evaluation* 68.2 (2011), pp. 90–104.

[KB06] Joachim Klein and Christel Baier. "Experiments with deterministic ω-automata for formulas of linear temporal logic". In: *Theoretical Computer Science* 363.2 (2006), pp. 182–195.

[KB07] Joachim Klein and Christel Baier. "On-the-Fly Stuttering in the Construction of Deterministic ω-Automata". In: *12th International Conference on Implementation and Application of Automata (CIAA)*. Vol. 4783. Lecture Notes in Computer Science. Springer, 2007, pp. 51–61.

[KE12] Jan Křetínský and Javier Esparza. "Deterministic Automata for the (F,G)-fragment of LTL". In: *24th International Conference on Computer Aided Verification (CAV)*. Vol. 7358. Lecture Notes in Computer Science. Springer, 2012, pp. 7–22.

[KG13] Jan Kretínský and Ruslán Ledesma Garza. "Rabinizer 2: Small Deterministic Automata for LTL\GU". In: *11th International Symposium on Automated Technology for Verification and Analysis (ATVA)*. Vol. 8172. Lecture Notes in Computer Science. Springer, 2013, pp. 446–450.

[Kie17] Stefan Kiefer. *Unambiguous Automata: From PSPACE to P*. Blog article from Stefan Kiefer. 2017. URL: `https://stekie.blogspot.de/2017/06/unambiguous-automata-from-pspace-to-p.html`.

[Kin17] Dileep Raghunath Kini. "Verification of Linear Time Properties For Finite Probabilistic Systems". PhD thesis. University of Illinois, 2017.

[Kle+14] Joachim Klein, David Müller, Christel Baier, and Sascha Klüppelholz. "Are Good-for-Games Automata Good for Probabilistic Model Checking?" In: *8th International Conference on Language and Automata Theory and Applications (LATA)*. Vol. 8370. Lecture Notes in Computer Science. Springer, 2014, pp. 453–465.

[Kle+17] Joachim Klein, Christel Baier, Philipp Chrszon, Marcus Daum, Clemens Dubslaff, Sascha Klüppelholz, Steffen Märcker, and David Müller. "Advances in probabilistic model checking with PRISM: variable reordering, quantiles and weak deterministic Büchi automata". In: *International Journal on Software Tools for Technology Transfer (STTT)* (2017), pp. 1–16.

[KM18] Denis Kuperberg and Anirban Majumdar. "Width of Non-deterministic Automata". In: *35th International Symposium on Theoretical Aspects of Computer Science (STACS)*. Vol. 96. Leibniz International Proceedings in Informatics. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[KNP04] Marta Zofia Kwiatkowska, Gethin Norman, and David Parker. "Probabilistic Symbolic Model Checking with PRISM: A Hybrid Approach". In: *International Journal on Software Tools for Technology Transfer (STTT)* 6(2) (2004), pp. 128–142.

[KNP11] Marta Zofia Kwiatkowska, Gethin Norman, and David Parker. "PRISM 4.0: Verification of Probabilistic Real-time Systems". In: *23th International Conference on Computer Aided Verification (CAV)*. Vol. 6806. Lecture Notes in Computer Science. Springer, 2011, pp. 585–591.

[KNP12] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. "The PRISM Benchmark Suite". In: *9th International Conference on Quantitative Evaluation of SysTems (QEST)*. IEEE Computer Society, 2012, pp. 203–204.

[KNS02] Marta Zofia Kwiatkowska, Gethin Norman, and Jeremy Sproston. "Probabilistic Model Checking of the IEEE 802.11 Wireless Local Area Network Protocol". In: *Second Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification, (PAPM-PROBMIV)*. Vol. 2399. Lecture Notes in Computer Science. Springer, 2002, pp. 169–187.

[KPV06] Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. "Safraless Compositional Synthesis". In: *18th International Conference on Computer Aided Verification, (CAV)*. Vol. 4144. Lecture Notes in Computer Science. Springer, 2006, pp. 31–44.

[KR11] Orna Kupferman and Adin Rosenberg. "The Blow-up in Translating LTL to Deterministic Automata". In: *6th International Workshop on Model Checking and Artificial Intelligence (MoChArt)*. Vol. 6572. Lecture Notes in Computer Science. Springer, 2011, pp. 85–94.

[KRS09] Jui-Yi Kao, Narad Rampersad, and Jeffrey Shallit. "On NFAs where all states are final, initial, or both". In: *Theoretical Computer Science* 410.47–49 (2009), pp. 5010–5021.

[KS15] Denis Kuperberg and Michał Skrzypczak. "On Determinisation of Good-for-Games Automata". In: *42nd International Colloquium on Automata, Languages, and Programming (ICALP)*. Vol. 9135. Lecture Notes in Computer Science. 2015, pp. 299–310.

[KV05] Orna Kupferman and Moshe Y. Vardi. "From Linear Time to Branching Time". In: *Transactions on Computational Logic* 6.2 (2005), pp. 273–294.

[KV15] Dileep Kini and Mahesh Viswanathan. "Limit Deterministic and Probabilistic Automata for $LTL \setminus \mathrm{G\ U}$". In: *21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 9035. Lecture Notes in Computer Science. Springer, 2015, pp. 628–642.

[KW08] Detlef Kähler and Thomas Wilke. "Complementation, Disambiguation, and Determinization of Büchi Automata Unified". In: *35th International Colloquium on Automata, Languages and Programming (ICALP)*. Vol. 5125. Lecture Notes in Computer Science. Springer, 2008, pp. 724–735.

[LDB10] Axel Legay, Benoît Delahaye, and Saddek Bensalem. "Statistical Model Checking: An Overview". In: *First International Conference on Runtime Verification (RV)*. Vol. 6418. Lecture Notes in Computer Science. Springer, 2010, pp. 122–135.

[Len13a] Rastislav Lenhardt. "Tulip: Model Checking Probabilistic Systems Using Expectation Maximisation Algorithm". In: *10th International Conference on Quantitative Evaluation of Systems (QEST)*. Vol. 8054. Lecture Notes in Computer Science. Springer, 2013, pp. 155–159.

[Len13b] Rastislav Lenhardt. "Two Variable and Linear Temporal Logic in Model Checking and Games". PhD thesis. University of Oxford, 2013.

[LH00] Timo Latvala and Keijo Heljanko. "Coping With Strong Fairness". In: *Fundamenta Informaticae* 43.1–4 (2000), pp. 1–19.

[Li+16a] Yong Li, Wanwei Liu, Andrea Turrini, Ernst Moritz Hahn, and Lijun Zhang. "An Efficient Synthesis Algorithm for Parametric Markov Chains Against Linear Time Properties". In: *Second International Symposium on Dependable Software Engineering: Theories, Tools, and Applications (SETTA)*. Vol. 9984. Lecture Notes in Computer Science. Springer, 2016, pp. 280–296.

[Li+16b] Yong Li, Lei Song, Yuan Feng, and Lijun Zhang. "Verify LTL with Fairness Assumptions Efficiently". In: *23rd International Symposium on Temporal Representation and Reasoning (TIME)*. IEEE Computer Society, 2016, pp. 41–50.

[Löd01] Christof Löding. "Efficient minimization of deterministic weak omega-automata". In: *Information Processing Letters* 79.3 (2001), pp. 105–109.

[LP18] Christof Löding and Anton Pirogov. "On Finitely Ambiguous Büchi Automata". In: *22nd International Conference on Developments in Language Theory (DLT)*. Vol. 11088. Lecture Notes in Computer Science. Springer, 2018, pp. 503–515.

[LP85] Orna Lichtenstein and Amir Pnueli. "Checking That Finite State Concurrent Programs Satisfy Their Linear Specification". In: *12th Symposium on Principles of Programming Languages (POPL)*. ACM, 1985, pp. 97–107.

[LPY95] Kim G. Larsen, Paul Pettersson, and Wang Yi. "Model-Checking for Real-Time Systems". In: *Fundamentals of Computation Theory*. Lecture Notes in Computer Science 965. 1995, pp. 62–88.

[LR81] Daniel Lehmann and Michael O. Rabin. "On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem". In: *8th Symposium on Principles of Programming Languages (POPL)*. ACM, 1981, pp. 133–138.

[LSS94] Nancy Lynch, Isaac Saias, and Roberto Segala. "Proving Time Bounds for Randomized Distributed Algorithms". In: *13th Symposium on Principles of Distributed Computing (PODC)*. ACM, 1994, pp. 314–323.

[LV15] Axel Legay and Mahesh Viswanathan. "Statistical model checking: challenges and perspectives". In: *STTT* 17.4 (2015), pp. 369–376.

[McN66] Robert McNaughton. "Testing and Generating Infinite Sequences by a Finite Automaton". In: *Information and Control* 9.5 (1966), pp. 521–530.

[MD15] Thibaud Michaud and Alexandre Duret-Lutz. "Practical Stutter-Invariance Checks for ω-Regular Languages". In: *22nd International SPIN Symposium on Model Checking of Software (SPIN)*. Vol. 9232. 2015, 2015, pp. 84–101.

[MH84] Satoru Miyano and Takeshi Hayashi. "Alternating Finite Automata on ω-Words". In: *Theoretical Computer Science* 32 (1984), pp. 321–330.

[Mic88] Max Michel. "Complementation is more difficult with automata on infinite words". In: *CNET*. 1988.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*. Vol. 92. Lecture Notes in Computer Science. Springer, 1980. ISBN: 3-540-10235-3.

[Mor10] Andreas Morgenstern. "Symbolic Controller Synthesis for LTL Specifications". PhD thesis. Technische Universität Kaiserslautern, 2010.

[Mos84] Andrzej W. Mostowski. "Regular expressions for infinite trees and a standard form of automata". In: *Fifth Symposium on Computation Theory (SCT)*. Vol. 208. Lecture Notes in Computer Science. Springer, 1984, pp. 157–168.

[MS08] Andreas Morgenstern and Klaus Schneider. "From LTL to Symbolically Represented Deterministic Automata". In: *9th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*. Vol. 4905. Lecture Notes in Computer Science. Springer, 2008, pp. 279–293.

[MS17] David Müller and Salomon Sickert. "LTL to Deterministic Emerson-Lei Automata". In: *8th International Symposium on Games, Automata, Logics and Formal Verification (GandALF)*. Vol. 256. Electronic Proceedings of Theoretical Computer Science. Open Publishing Association, 2017, pp. 180–194.

[MS95] David E. Muller and Paul E. Schupp. "Simulating Alternating Tree Automata by Nondeterministic Automata: New Results and New Proofs of the Theorems of Rabin, McNaughton and Safra". In: *Theoretical Computer Science* 141.1&2 (1995), pp. 69–107.

[MT98] Christoph Meinel and Thorsten Theobald. *Algorithmen und Datenstrukturen im VLSI-Design*. Springer, 1998.

[Mül18] David Müller. Website with additional material. 2018. URL: `https://wwwtcs.inf.tu-dresden.de/~david/phd/`.

[Mul63] David E. Muller. "Infinite sequences and finite machines". In: *4th Symposium on Switching Circuit Theory and Logical Design*. IEEE computer society, 1963, pp. 3–16.

[NK14] Tobias Nipkow and Gerwin Klein. *Concrete Semantics - With Isabelle/HOL*. Springer, 2014. ISBN: 978-3-319-10541-3.

[Oss10] Jörn Ossowski. "JINC: A Multi-Threaded Library for Higher-Order Weighted Decision Diagram Manipulation". PhD thesis. University of Bonn, 2010.

[Par02] David Parker. "Implementation of Symbolic Model Checking for Probabilistic Systems". PhD thesis. University of Birmingham, 2002.

[Par81] David Park. "Concurrency and automata on infinite sequences". In: *Fifth GI-Conference on Theoretical Computer Science*. Springer, 1981, pp. 167–183.

[Pit07] Nir Piterman. "From Nondeterministic Büchi and Streett Automata to Deterministic Parity Automata". In: *Logical Methods in Computer Science* 3(3:5) (2007), pp. 1–21.

[Pnu77] Amir Pnueli. "The Temporal Logic of Programs". In: *18th Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 1977, pp. 46–57.

[PP06] Nir Piterman and Amir Pnueli. "Faster Solutions of Rabin and Streett Games". In: *21th IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society, 2006, pp. 275–284.

[PPS06] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. "Synthesis of Reactive(1) Designs". In: *7th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*. Vol. 3855. Lecture Notes in Computer Science. Springer, 2006, pp. 364–380.

[Put94] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[PZ86a] Amir Pnueli and Lenore D. Zuck. "Probabilistic Verification by Tableaux". In: *First Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society, 1986, pp. 322–331.

[PZ86b] Amir Pnueli and Lenore D. Zuck. "Verification of Multiprocess Probabilistic Protocols". In: *Distributed Computing* 1.1 (1986), pp. 53–72.

[Rab18] Alexander Rabinovich. "Complementation of Finitely Ambiguous Büchi Automata". In: *22nd International Conference on Developments in Language Theory (DLT)*. Vol. 11088. Lecture Notes in Computer Science. Springer, 2018, pp. 541–552.

[Rab72] Michael O. Rabin. *Automata on Infinite Objects and Church's Problem*. American Mathematical Society, 1972.

[Red12] Roman R. Redziejowski. "An Improved Construction of Deterministic Omega-automaton Using Derivatives". In: *Fundamenta Informaticae* 119.3-4 (2012), pp. 393–406.

[RS59] Michael O. Rabin and Dana S. Scott. "Finite Automata and Their Decision Problems". In: *IBM Journal of Research and Development* 3.2 (1959), pp. 114–125.

[RT96] Monika Rauch Henzinger and Jan Arne Telle. "Faster Algorithms for the Nonemptiness of Streett Automata and for Communication Protocol Pruning." In: *5th Scandinavian Workshop on Algorithm Theory (SWAT)*. Vol. 1097. Lecture Notes in Computer Science. Springer-Verlag, 1996, pp. 16–27.

[Rud93] Richard Rudell. "Dynamic Variable Ordering for Ordered Binary Decision Diagrams". In: *11th International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1993, pp. 42–47.

[RV10] Kristin Y. Rozier and Moshe Y. Vardi. "LTL Satisfiability Checking". In: *International Journal on Software Tools for Technology Transfer (STTT)* 12.2 (2010), pp. 123–137.

[Saf88] Shmuel Safra. "On The Complexity of ω-Automata". In: *29th Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 1988, pp. 319–327.

[SB00] Fabio Somenzi and Roderick Bloem. "Efficient Büchi Automata from LTL Formulae". In: *12th International Conference on Computer Aided Verification (CAV)*. Vol. 1855. Lecture Notes in Computer Science. Springer, 2000, pp. 248–263.

[SC85] A. Prasad Sistla and Edmund M. Clarke. "The Complexity of Propositional Linear Temporal Logics". In: *Journal of the ACM* 32.3 (1985), pp. 733–749.

[Sch09] Sven Schewe. "Tighter Bounds for the Determinisation of Büchi Automata". In: *12th Conference on Foundations of Software Science and Computational Structures (FOSSACS)*. Vol. 5504. Lecture Notes in Computer Science. Springer, 2009, pp. 167–181.

[Sch10] Sven Schewe. "Beyond Hyper-Minimisation – Minimising DBAs and DPAs is NP-Complete". In: *30th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Vol. 8. Leibniz International Proceedings in Informatics. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010, pp. 400–411.

[SF07] Sven Schewe and Bernd Finkbeiner. "Bounded Synthesis". In: *Fifth Internation Symposium on Automated Technology for Verification and Analysis (ATVA)*. Vol. 4762. Lecture Notes in Computer Science. Springer, 2007, pp. 474–488.

[SH85] Richard E. Stearns and Harry B. Hunt. "On the equivalence and containment problem for unambiguous regular expressions, grammars, and automata". In: *SIAM Journal on Computing* (1985), pp. 598–611.

[Sic+16] Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Křetínský. "Limit-Deterministic Büchi Automata for Linear Temporal Logic". In: *28th International Conference on Computer Aided Verification (CAV)*. Vol. 9780. Lecture Notes in Computer Science. Springer, 2016, pp. 312–332.

[SK16] Salomon Sickert and Jan Křetínský. "MoChiBA: Probabilistic LTL Model Checking Using Limit-Deterministic Büchi Automata". In: *14th International Symposium on Automated Technology for Verification and Analysis (ATVA)*. Vol. 9938. Lecture Notes in Computer Science. Springer, 2016, pp. 130–137.

[Str82] Robert S. Streett. "Propositional Dynamic Logic of Looping and Converse Is Elementarily Decidable". In: *Information and Control* 54.1/2 (1982), pp. 121–141.

[SV89] Shmuel Safra Safra and Moshe Y. Vardi. "On Ω-automata and Temporal Logic". In: *21th Symposium on Theory of Computing (STOC)*. ACM, 1989.

[Tho97] Wolfgang Thomas. "Languages, Automata, and Logic". In: *Handbook of Formal Languages*. Vol. 3. Springer, 1997, pp. 389–455.

[Tra62] Boris Avraamovich Trakhtenbrot. "Finite Automata and The Logic of One-Place Predicates". In: *Siberian Mathematical Journal* 3 (1962), pp. 103–131.

[Tse68] Gregory S. Tseitin. "On the Complexity of Derivations in the Propositional Calculus". In: *Studies in Mathematics and Mathematical Logic* II (1968), pp. 115–125.

[Tur17] Andrea Turrini. *ePMC is publicly available*. Probabilistic Model Checking Team @ ISCAS. 2017. URL: `http://iscasmc.ios.ac.cn/?p=1241` (visited on 08/08/2018).

[Var85] Moshe Y. Vardi. "Automatic Verification of Probabilistic Concurrent Finite-State Programs". In: *26th IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 1985, pp. 327–338.

[Var99] Moshe Y. Vardi. "Probabilistic Linear-Time Model Checking: An Overview of the Automata-Theoretic Approach". In: *Fifth International AMAST Workshop on Formal Methods for Real-Time and Probabilistic Systems (ARTS)*. Vol. 1601. Lecture Notes in Computer Science. Springer, 1999, pp. 265–276.

[VW86] Moshe Y. Vardi and Pierre Wolper. "An Automata-Theoretic Approach to Automatic Program Verification". In: *1st Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society, 1986, pp. 332–344.

[Weg00] Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000.

[WVS83] Pierre Wolper, Moshe Y. Vardi, and A. Prasad Sistla. "Reasoning about Infinite Computation Paths (Extended Abstract)". In: *24th Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 1983, pp. 185–194.

[Zim13] Martin Zimmermann. "Optimal bounds in parametric LTL games". In: *Theoretical Computer Science* 493 (2013), pp. 30–45.